

# String Commands

The string interface enables application control of media devices using textual string commands. String commands are passed to the media control interface through the [mciSendString](#) function. Return information from string commands is converted to string format and returned in the *pszReturnString* parameter of [mciSendString](#).

Not all functions available through the procedural interface, [mciSendCommand](#), are available through the string interface. In general, operations that return complex data structures, such as a CD table of contents, are available only through the procedural interface. Operations that cause asynchronous responses to be generated, such as cue point and position advise, can be invoked from the string interface; however, their responses are returned to window procedures.

The keywords WAIT and NOTIFY are global keywords and are available for all commands except some system commands. As with the procedural interface, the default time base is MMTIME. The multimedia string parser is case insensitive.

## Command Format

The string format is:

```
COMMAND      object
              keywords      WAIT
                              NOTIFY
```

where

```
object        = device type | device name | filename | alias
keywords      = command-specific keywords
WAIT | NOTIFY = standard OS/2 multimedia definitions
```

The object associated with a media control interface command can be one of the following:

- **Device type** - The default device of a given type. The possible types of controllable devices include the following:

Device	Description
videotape	Videotape player or recorder
videodisc	Videodisc player
cdaudio	CD-ROM device that supports standard compact disc playback
waveaudio	Device that supports digital audio files
sequencer	Device that supports MIDI files
digitalvideo	Device that supports audio/video files, either hardware-assisted or software motion video only

If you have multiple devices of the same type, the Multimedia Setup program allows you to decide which device should be the default for that type.

- **Device name** - A name of a particular device. Device names are of the form *DevicetypeNN*, where *Devicetype* is one of the device types given above, and *NN* is a value (01, 02,...) indicating which device of that type is to be controlled.
- **Filename** - The name of a file to be opened or controlled. When a filename is opened, OS/2 multimedia first examines the file's extension, then its type, to determine which device is associated to the file.
- **Alias** - A string that was specified on a previous OPEN command. This string can then be used as the object name in subsequent commands.

The only exception to the above command format is [MASTERAUDIO](#), which does not require an object associated with the command. The format for [MASTERAUDIO](#) is:

```
MASTERAUDIO  keywords
              WAIT
```

-----

## How to Read the Syntax Diagram

The syntax diagram shows you how to specify a command so that the multimedia string parser can correctly interpret what you type. Read the syntax diagram from left to right and from top to bottom, following the horizontal baseline (the main path). The command name and items

required to make the command work appear on the baseline; the items below the baseline are optional.

A line *ending* with an arrowhead means that the command syntax is continued. A line *starting* with an arrowhead means that the syntax is continued from the previous line. A vertical bar marks the end of the command syntax.

Command names are often followed by required or optional *keywords*, which affect the result of the command. Variables are represented in lowercase and must be replaced with a valid name or value you specify. In the following example, **object**, **devicealias**, and **devicetype** are variables. You must include any punctuation, such as parentheses or commas, that are shown in the diagram.

```

OPEN      object
          ALIAS devicealias
          SHAREABLE
          TYPE devicetype
          WAIT
          NOTIFY

```

In the OPEN command shown above, **object** is required, the **ALIAS**, **SHAREABLE**, and **TYPE** keywords are optional, and the **WAIT** and **NOTIFY** keywords are also optional.

## Specifying Items Once in Any Order

A stack of keywords with a return arrow above the main path indicates that you can specify one or more keywords in any order, but you can specify each keyword only once.

```
COPY      object
          FROM pos      WAIT
          TO pos        NOTIFY
```

## Specifying One Item from a Stack

A stack of keywords with no return arrow means that you cannot specify more than one keyword from the stack.

SEEK	object	TO pos	
		TO START	WAIT
		TO END	NOTIFY

# System Commands

System commands are interpreted directly by the media device manager (MDM), and are not passed to media control interface drivers. The following commands are system commands:

- ACQUIRE
- CONNECTION
- CONNECTORINFO
- DEFAULTCONNECTION
- GROUP
- MASTERAUDIO
- RELEASE
- SYSINFO

# ACQUIRE

## ACQUIRE (System Command) - Example

```
acquire digitalvideo exclusive wait
```

---

## ACQUIRE (System Command) - Purpose

The ACQUIRE command acquires use of the physical resources for the device. The EXCLUSIVE and EXCLUSIVE INSTANCE keywords cannot be used together.

---

## ACQUIRE (System Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## ACQUIRE (System Command) Keyword - EXCLUSIVE

### **EXCLUSIVE**

Acquires the physical resource for exclusive use. If the resource is not available, MCIERR\_DEVICE\_IN\_USE is returned. Exclusive use of a device can be released with the [RELEASE](#) system command.

---

## ACQUIRE (System Command) Keyword - EXCLUSIVE INSTANCE

### **EXCLUSIVE INSTANCE**

Acquires the device such that whether being used or not, it cannot be made inactive by another request.

---

## ACQUIRE (System Command) Keyword - QUEUE

**QUEUE**

Queues ACQUIRE command to be executed when device resources become available.

---

## ACQUIRE (System Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## ACQUIRE (System Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## ACQUIRE (System Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**EXCLUSIVE**

Acquires the physical resource for exclusive use. If the resource is not available, MCIERR\_DEVICE\_IN\_USE is returned. Exclusive use of a device can be released with the [RELEASE](#) system command.

**EXCLUSIVE INSTANCE**

Acquires the device such that whether being used or not, it cannot be made inactive by another request.

**QUEUE**

Queues ACQUIRE command to be executed when device resources become available.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## ACQUIRE (System Command) - Syntax Diagram

ACQUIRE      object      EXCLUSIVE      QUEUE  
EXCLUSIVE INSTANCE  
  
WAIT  
NOTIFY

Examples

-----

## ACQUIRE (System Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

-----

## CONNECTION

-----

## CONNECTION (System Command) - Example

```
open waveaudio alias wave shareable wait
```

The following command returns the alias name of the connected device ("ampmix").

```
connection wave query type wave stream alias ampmix wait
```

-----

## CONNECTION (System Command) - Purpose

The CONNECTION command returns information about the device context connections.

-----

## CONNECTION (System Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## CONNECTION (System Command) Keyword - QUERY

**QUERY**

Queries the connection as defined by the NUMBER and TYPE item. If a connection exists, the alias of the connected device is returned. If no alias is defined, then a null string is returned. If QUERY and ALIAS are both specified, then the specified alias name is returned and assigned, if possible. See the ALIAS keyword for more information on possible errors.

-----

## CONNECTION (System Command) Keyword - NUMBER connector\_number

**NUMBER connector\_number**

Indicates the connector number to which this query applies. If this item is omitted, then the first connector is assumed. If the TYPE keyword is included, then the connector number is interpreted as a relative offset within the specified connector type.

**Note:** This keyword is used in conjunction with the QUERY keyword.

-----

## CONNECTION (System Command) Keyword - TYPE connector\_type

**TYPE connector\_type**

The type of connector to which the requested action applies. The following connector types are defined:

**Note:** This keyword is used in conjunction with the QUERY keyword.

MIDI stream

Digital input or output for a sequencer device. This data is typically streamed to an amplifier mixer device.

CD stream

Digital output for a CD audio device capable of reading the data directly off of a disk. The data is typically streamed to an amplifier mixer device.

wave stream

Digital input or output for a waveform audio device. The data is typically streamed to an amplifier mixer device.

XA stream

Digital output for a CD-ROM/XA device. The data is typically streamed to an amplifier mixer device.

amp stream	Digital input or output for an amplifier mixer device.
headphones	The connector on the device which is labeled or is typically used to attach headphones to the device.
speakers	The connector on the device which is labeled or is typically used to attach speakers to the device.
microphone	The connector on the device which is labeled or is typically used to attach a microphone to the device.
line in	The connector on the device which is labeled or is typically used to provide line level input to the device.
line out	The connector on the device which is labeled or is typically used to provide line level output from the device.
video in	The connector on the device which is labeled or is typically used to provide video input to the device.
video out	The connector on the device which is labeled or is typically used to provide video output from the device.

## CONNECTION (System Command) Keyword - ALIAS device\_alias

### ALIAS device\_alias

Defines an alias for the device connected to the specified connector. If the alias already exists for another device the error MCIERR\_DUPLICATE\_ALIAS is returned. If the device connected to already has an alias the error MCIERR\_CANNOT\_ADD\_ALIAS is returned.

**Note:** This keyword is used in conjunction with the QUERY keyword.

## CONNECTION (System Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

## CONNECTION (System Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

## CONNECTION (System Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**QUERY**

Queries the connection as defined by the NUMBER and TYPE item. If a connection exists, the alias of the connected device is returned. If no alias is defined, then a null string is returned. If QUERY and ALIAS are both specified, then the specified alias name is returned and assigned, if possible. See the ALIAS keyword for more information on possible errors.

**NUMBER connector\_number**

Indicates the connector number to which this query applies. If this item is omitted, then the first connector is assumed. If the TYPE keyword is included, then the connector number is interpreted as a relative offset within the specified connector type.

**Note:** This keyword is used in conjunction with the QUERY keyword.

**TYPE connector\_type**

The type of connector to which the requested action applies. The following connector types are defined:

**Note:** This keyword is used in conjunction with the QUERY keyword.

MIDI stream	Digital input or output for a sequencer device. This data is typically streamed to an amplifier mixer device.
CD stream	Digital output for a CD audio device capable of reading the data directly off of a disk. The data is typically streamed to an amplifier mixer device.
wave stream	Digital input or output for a waveform audio device. The data is typically streamed to an amplifier mixer device.
XA stream	Digital output for a CD-ROM/XA device. The data is typically streamed to an amplifier mixer device.
amp stream	Digital input or output for an amplifier mixer device.
headphones	The connector on the device which is labeled or is typically used to attach headphones to the device.
speakers	The connector on the device which is labeled or is typically used to attach speakers to the device.
microphone	The connector on the device which is labeled or is typically used to attach a microphone to the device.
line in	The connector on the device which is labeled or is typically used to provide line level input to the device.
line out	The connector on the device which is labeled or is typically used to provide line level output from the device.
video in	The connector on the device which is labeled or is typically used to provide video input to the device.
video out	The connector on the device which is labeled or is typically used to provide video output from the device.

**ALIAS device\_alias**

Defines an alias for the device connected to the specified connector. If the alias already exists for another device the error MCIERR\_DUPLICATE\_ALIAS is returned. If the device connected to already has an alias the error MCIERR\_CANNOT\_ADD\_ALIAS is returned.

**Note:** This keyword is used in conjunction with the QUERY keyword.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the



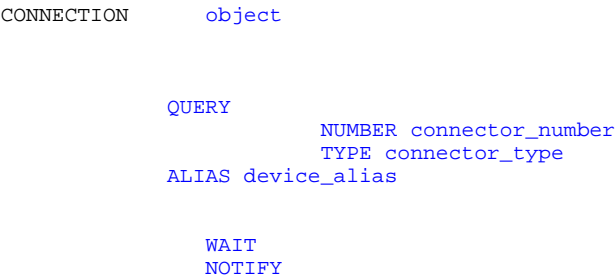
application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# CONNECTION (System Command) - Syntax Diagram



---

# CONNECTION (System Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

# CONNECTORINFO

---

# CONNECTORINFO (System Command) - Example

The following command returns "wave stream".

```
connectorinfo digitalvideo typeof number 1 wait
```

---

## CONNECTORINFO (System Command) - Purpose

The CONNECTORINFO command returns information about the connectors on a device.

---

## CONNECTORINFO (System Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CONNECTORINFO (System Command) Keyword - ENUMERATE

### **ENUMERATE**

Returns the number of connectors of the specified type. If no TYPE keyword is specified, then the total number of connectors for the device is returned.

---

## CONNECTORINFO (System Command) Keyword - TYPE connector\_type

### **TYPE connector\_type**

Indicates the type of connector to which the query applies. The connector types are defined for each device. See the TYPE keyword for [CONNECTION](#) for a list of supported connector types.

---

## CONNECTORINFO (System Command) Keyword - TYPEOF

### **TYPEOF**

Returns the connector type of the specified connector. Use of this option requires that the NUMBER keyword must also be specified.

---

## CONNECTORINFO (System Command) Keyword - NUMBER

## connector\_number

### **NUMBER connector\_number**

Indicates the connector number to which this query applies.

---

## CONNECTORINFO (System Command) Keyword - CAN CONNECT TO connector\_type

### **CAN CONNECT TO connector\_type**

Returns true if this connector type is compatible with the connector type of the specified connector; that is, results in a valid connection. Use of this option requires that the TYPE keyword must also be specified.

---

## CONNECTORINFO (System Command) Keyword - TYPE connector\_type

### **TYPE connector\_type**

Indicates the type of connector to which the query applies. The connector types are defined for each device. See the TYPE keyword for [CONNECTION](#) for a list of supported connector types.

---

## CONNECTORINFO (System Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified to receive return string information.

---

## CONNECTORINFO (System Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CONNECTORINFO (System Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**ENUMERATE**

Returns the number of connectors of the specified type. If no TYPE keyword is specified, then the total number of connectors for the device is returned.

**TYPE connector\_type**

Indicates the type of connector to which the query applies. The connector types are defined for each device. See the TYPE keyword for [CONNECTION](#) for a list of supported connector types.

**TYPEOF**

Returns the connector type of the specified connector. Use of this option requires that the NUMBER keyword must also be specified.

**NUMBER connector\_number**

Indicates the connector number to which this query applies.

**CAN CONNECT TO connector\_type**

Returns true if this connector type is compatible with the connector type of the specified connector; that is, results in a valid connection. Use of this option requires that the TYPE keyword must also be specified.

**TYPE connector\_type**

Indicates the type of connector to which the query applies. The connector types are defined for each device. See the TYPE keyword for [CONNECTION](#) for a list of supported connector types.

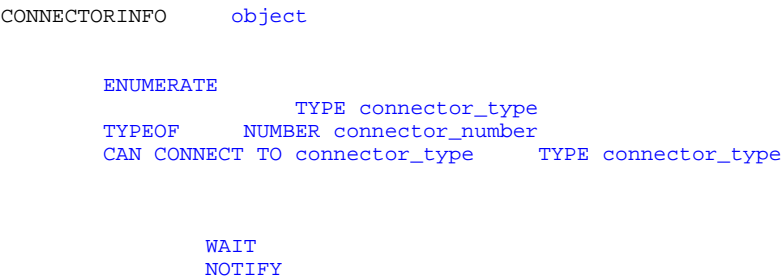
**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified to receive return string information.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

CONNECTORINFO (System Command) - Syntax Diagram



CONNECTORINFO (System Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## DEFAULTCONNECTION

---

### DEFAULTCONNECTION (System Command) - Example

The following command returns "ampmix01 ampmix 1"

```
defaultconnection digitalvideo query wait
```

---

### DEFAULTCONNECTION (System Command) - Purpose

The DEFAULTCONNECTION command makes, breaks, or queries a default connection.

---

### DEFAULTCONNECTION (System Command) Keyword - object

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

### DEFAULTCONNECTION (System Command) Keyword - MAKE TO devicename

#### **MAKE TO devicename**

Establish a connection. The devicename is necessary for connection to be established. Use of the MAKE TO keyword also requires

the TONUMBER and/or TOTYPE keyword.

-----

## DEFAULTCONNECTION (System Command) Keyword - TYPE connector\_type

### **TYPE connector\_type**

Indicates the connector type.

-----

## DEFAULTCONNECTION (System Command) Keyword - NUMBER connector\_number

### **NUMBER connector\_number**

Indicates the connector number to which the action applies. If this item is omitted, then the first connector is assumed. If the TYPE keyword is included, then the connector number is interpreted as a relative offset within the specified connector type.

-----

## DEFAULTCONNECTION (System Command) Keyword - TOTYPE connector\_type

### **TOTYPE connector\_type**

Indicates the type of connector on the target device.

-----

## DEFAULTCONNECTION (System Command) Keyword - TONUMBER connector\_number

### **TONUMBER connector\_number**

Indicates the connector number on the target device during a MAKE action. If this item is omitted, the first connector is assumed. If the TOTYPE keyword is included, then the connector number is interpreted as a relative offset within the specified connector type.

-----

## DEFAULTCONNECTION (System Command) Keyword - BREAK

**BREAK**

Delete a connection. If the BREAK keyword is specified, the TYPE keyword is also required.

---

## DEFAULTCONNECTION (System Command) Keyword - TYPE connector\_type

**TYPE connector\_type**

Indicates the connector type.

---

## DEFAULTCONNECTION (System Command) Keyword - QUERY

**QUERY**

Query a connection. Returns the devicename, connector\_type, and connector\_number.

---

## DEFAULTCONNECTION (System Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT flag must be specified to receive return string information.

---

## DEFAULTCONNECTION (System Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## DEFAULTCONNECTION (System Command) - Keywords

object





## Examples

---

# DEFAULTCONNECTION (System Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## GROUP

---

## GROUP (System Command) - Example

```
group mygroup make (cd wave1) wait
```

---

## GROUP (System Command) - Purpose

The GROUP command allows an application to control multiple devices as a unit, or group. Once the group is established, the application can control all the devices in the group with a single command. If CLOSE is issued directly to an instance in the group, the instance is deleted from the group.

All other commands addressed to the group must include the NOTIFY flag.

---

## GROUP (System Command) Keyword - groupname

### **groupname**

Refers to a group using a name instead of a group ID. A value for this variable must be specified with the MAKE keyword.

**Note:** In use, the variable **groupname** is the same as an alias. Although the ALIAS keyword is not used, all rules related to ALIAS apply to the group name.

---

## GROUP (System Command) Keyword - MAKE

**MAKE**

Specifies creation of a group by tying several instances together. Once "grouped", the instances can be controlled by the application with one command.

The MAKE keyword requires values for the variables **groupname** and **(i1 i2 i3)**.

---

## GROUP (System Command) Keyword - (i1 i2 i3)

**(i1 i2 i3)**

Refers to the device instances that make up the group. Device instances can be identified using aliases, device types, or filenames-the same identifiers used when the devices were opened. A value for this variable must be specified with the MAKE keyword.

---

## GROUP (System Command) Keyword - NOPIECEMEAL

**NOPIECEMEAL**

Specifies that the group is to be processed as a whole rather than as separate parts. If one instance becomes inactive, all instances become inactive. This keyword is used only with MAKE.

---

## GROUP (System Command) Keyword - DELETE

**DELETE**

Terminates an existing group by disassociating instances that formed the group. No other keywords, except WAIT or NOTIFY, can be used with DELETE.

---

## GROUP (System Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## GROUP (System Command) Keyword - NOTIFY

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an `MM_MCINOTIFY` message is sent to the application window procedure.

-----

## GROUP (System Command) - Keywords

#### groupname

Refers to a group using a name instead of a group ID. A value for this variable must be specified with the MAKE keyword.

**Note:** In use, the variable **groupname** is the same as an alias. Although the ALIAS keyword is not used, all rules related to ALIAS apply to the group name.

#### MAKE

Specifies creation of a group by tying several instances together. Once "grouped", the instances can be controlled by the application with one command.

The MAKE keyword requires values for the variables **groupname** and **(i1 i2 i3)**.

#### (i1 i2 i3)

Refers to the device instances that make up the group. Device instances can be identified using aliases, device types, or filenames-the same identifiers used when the devices were opened. A value for this variable must be specified with the MAKE keyword.

#### NOPIECEMEAL

Specifies that the group is to be processed as a whole rather than as separate parts. If one instance becomes inactive, all instances become inactive. This keyword is used only with MAKE.

#### DELETE

Terminates an existing group by disassociating instances that formed the group. No other keywords, except WAIT or NOTIFY, can be used with DELETE.

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an `MM_MCINOTIFY` message is sent to the application window procedure.

-----

## GROUP (System Command) - Syntax Diagram

```
GROUP      groupname      MAKE      (i1 i2 i3)      NOPIECEMEAL
                                     DELETE
                                     WAIT
                                     NOTIFY
```

Examples

-----

# GROUP (System Command) - Topics

Select an item:

[Purpose](#)

[Syntax Diagram](#)

[Keywords](#)

[Example](#)

[Glossary](#)

---

## MASTERAUDIO

---

## MASTERAUDIO (System Command) - Example

```
masteraudio query volume wait
```

---

## MASTERAUDIO (System Command) - Purpose

The MASTERAUDIO command sets the system master audio setting for all audio devices in the system. The WAIT flag must be specified to get string return information for queries.

This command is used by the OS/2 multimedia Volume Control application to control system-wide audio parameters based on user preference. Applications should take special care when using MASTERAUDIO, as it results in a system-wide change. Typically, applications control the volume only within an application.

---

## MASTERAUDIO (System Command) Keyword - VOLUME level

### **VOLUME level**

Sets the system-wide master volume to the level specified as a percentage.

---

## MASTERAUDIO (System Command) Keyword - QUERY VOLUME

#### **QUERY VOLUME**

Returns the current application controlled master volume level. 0-100 is returned.

-----

## MASTERAUDIO (System Command) Keyword - QUERY HEADPHONES

#### **QUERY HEADPHONES**

Queries the system-wide headphone setting. ON or OFF is returned.

-----

## MASTERAUDIO (System Command) Keyword - QUERY SPEAKERS

#### **QUERY SPEAKERS**

Queries the system-wide speaker setting. ON or OFF is returned.

-----

## MASTERAUDIO (System Command) Keyword - WAIT

#### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified to receive return string information.

-----

## MASTERAUDIO (System Command) - Keywords

#### **VOLUME level**

Sets the system-wide master volume to the level specified as a percentage.

#### **QUERY VOLUME**

Returns the current application controlled master volume level. 0-100 is returned.

#### **QUERY HEADPHONES**

Queries the system-wide headphone setting. ON or OFF is returned.

#### **QUERY SPEAKERS**

Queries the system-wide speaker setting. ON or OFF is returned.

#### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified to receive return string information.

-----

# MASTERAUDIO (System Command) - Syntax Diagram

MASTERAUDIO

VOLUME level  
QUERY VOLUME  
QUERY HEADPHONES  
QUERY SPEAKERS

WAIT



---

## MASTERAUDIO (System Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

## RELEASE

---

## RELEASE (System Command) - Example

```
release digitalvideo return resource wait
```

---

## RELEASE (System Command) - Purpose

The RELEASE command releases exclusive use of the physical resources by the device context.

---

## RELEASE (System Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## RELEASE (System Command) Keyword - RETURN RESOURCE

**RETURN RESOURCE**

Returns resource for any instance that has requested and is waiting for the resource. If the resource is not requested by another instance, it is left active. If resource used is not required by any other instance, it is left active.

-----

## RELEASE (System Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## RELEASE (System Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## RELEASE (System Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**RETURN RESOURCE**

Returns resource for any instance that has requested and is waiting for the resource. If the resource is not requested by another instance, it is left active. If resource used is not required by any other instance, it is left active.

**WAIT**

**NOTIFY**

-----

### Examples

-----

-----

---

## SYSINFO (System Command) - Purpose



The SYSINFO command obtains information about the devices installed in the system. This command also accepts ALL as the device name. If ALL is used, system information is returned for all devices in the system.

-----

## SYSINFO (System Command) Keyword - object

**object**  
Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## SYSINFO (System Command) Keyword - INSTALLNAME

**INSTALLNAME**  
Returns the name that was used to install the device.

-----

## SYSINFO (System Command) Keyword - QUANTITY

**QUANTITY**  
Returns the number of media control interface devices of the type specified by the device name. The device name must be a standard media control interface device type. Any digits after the name are ignored. The special device name **all** returns the total number of media control interface devices in the system.

-----

## SYSINFO (System Command) Keyword - QUANTITY OPEN

**QUANTITY OPEN**  
Returns the number of open media control interface devices of the type specified by the device name. The device name must be a standard media control interface device type. Any digits after the name are ignored. The special device name **all** returns the total number of media control interface devices in the system.

-----

## SYSINFO (System Command) Keyword - NAME number

**NAME number**

Returns the name of a media control interface device. The **number** (ordinal) ranges from 1 to the number of devices of that type. If **all** is specified for the device name, then the number must still be provided, but it is ignored.

-----

## SYSINFO (System Command) Keyword - NAME number OPEN

### NAME number OPEN

Returns the name of an open media control interface device. The **number** (ordinal) ranges from 1 to the number of devices of that type. If **all** is specified for the device name, then the number must still be provided, but it is ignored and all open device names are returned.

-----

## SYSINFO (System Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## SYSINFO (System Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### INSTALLNAME

Returns the name that was used to install the device.

### QUANTITY

Returns the number of media control interface devices of the type specified by the device name. The device name must be a standard media control interface device type. Any digits after the name are ignored. The special device name **all** returns the total number of media control interface devices in the system.

### QUANTITY OPEN

Returns the number of open media control interface devices of the type specified by the device name. The device name must be a standard media control interface device type. Any digits after the name are ignored. The special device name **all** returns the total number of media control interface devices in the system.

### NAME number

Returns the name of a media control interface device. The **number** (ordinal) ranges from 1 to the number of devices of that type. If **all** is specified for the device name, then the number must still be provided, but it is ignored.

### NAME number OPEN

Returns the name of an open media control interface device. The **number** (ordinal) ranges from 1 to the number of devices of that type. If **all** is specified for the device name, then the number must still be provided, but it is ignored and all open device names are returned.

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## SYSINFO (System Command) - Syntax Diagram

```
SYSINFO      object      INSTALLNAME
                QUANTITY      WAIT
                QUANTITY OPEN
                NAME number
                NAME number OPEN
```



---

## SYSINFO (System Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

## Required Commands

Required commands are standard commands that are recognized by *all* MCI devices. Additional options can be added to extend these commands, however an MCI device must still support the required options. The following commands are required commands:

- [CAPABILITY](#)
- [CLOSE](#)
- [INFO](#)
- [OPEN](#)
- [STATUS](#)

---

## CAPABILITY

---

## CAPABILITY (Required Command) - Example

capability waveaudio01 can record wait

---

## CAPABILITY (Required Command) - Purpose

The CAPABILITY command requests the information about a particular capability of a device.

---

## CAPABILITY (Required Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CAPABILITY (Required Command) Keyword - CAN EJECT

### **CAN EJECT**

Returns TRUE if the device can eject the media.

---

## CAPABILITY (Required Command) Keyword - CAN PLAY

### **CAN PLAY**

Returns TRUE if the device can play.

---

## CAPABILITY (Required Command) Keyword - CAN RECORD

### **CAN RECORD**

Returns TRUE if the device supports recording.

---

## CAPABILITY (Required Command) Keyword - CAN SAVE

**CAN SAVE**

Returns TRUE if the device can save data.

-----

## CAPABILITY (Required Command) Keyword - CAN LOCKEJECT

**CAN LOCKEJECT**

Returns TRUE if the device can disable the manual ejection of the media.

-----

## CAPABILITY (Required Command) Keyword - CAN SETVOLUME

**CAN SETVOLUME**

Returns TRUE if the device supports software control of volume level.

-----

## CAPABILITY (Required Command) Keyword - COMPOUND DEVICE

**COMPOUND DEVICE**

Returns TRUE if the device requires an element name.

-----

## CAPABILITY (Required Command) Keyword - DEVICE TYPE

**DEVICE TYPE**

Returns one of the following:

- ampmix
- cdaudio
- cdx
- digitalvideo
- overlay
- sequencer
- videodisc
- waveaudio
- other

---

# CAPABILITY (Required Command) Keyword - HAS AUDIO

**HAS AUDIO**  
Returns TRUE if the device supports audio playback.

---

# CAPABILITY (Required Command) Keyword - HAS VIDEO

**HAS VIDEO**  
Returns TRUE if the device supports video playback.

---

# CAPABILITY (Required Command) Keyword - MESSAGE command

**MESSAGE command**  
Returns TRUE if the device supports the command specified by **command**. Following are the commands you can query:

- |             |                   |
|-------------|-------------------|
| ACQUIRE     | RECORD            |
| CAPABILITY  | RELEASE           |
| CLOSE       | RESUME            |
| CONNECTION  | SAVE              |
| CONNECTOR   | SEEK              |
| CUE         | SET               |
| ESCAPE      | SETCUEPOINT       |
| GROUP       | SETPOSITIONADVISE |
| INFO        | SPIN              |
| LOAD        | STATUS            |
| MASTERAUDIO | STEP              |
| OPEN        | STOP              |
| PAUSE       | SYSINFO           |
| PLAY        |                   |

---

# CAPABILITY (Required Command) Keyword - PREROLL TIME

**PREROLL TIME**  
Returns the deterministic or maximum notified preroll time in MMTIME units. A value of 0 for a notified preroll device indicates the preroll time is not bounded.

---

# CAPABILITY (Required Command) Keyword - PREROLL

# TYPE

## PREROLL TYPE

Returns the preroll characteristics of the device. Returns **notified** if the preroll time for the device is variable. Returns **deterministic** if the prerolled time for the device is fixed. Returns **none** if the device does not support preroll.

-----

# CAPABILITY (Required Command) Keyword - USES FILES

## USES FILES

Returns TRUE if the device requires a file path name.

-----

# CAPABILITY (Required Command) Keyword - WAIT

## WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

-----

# CAPABILITY (Required Command) Keyword - NOTIFY

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# CAPABILITY (Required Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

While other capabilities are defined for specific devices and device types, the following keywords can always be specified:

## CAN EJECT

Returns TRUE if the device can eject the media.

## CAN PLAY

Returns TRUE if the device can play.

#### CAN RECORD

Returns TRUE if the device supports recording.

#### CAN SAVE

Returns TRUE if the device can save data.

#### CAN LOCKEJECT

Returns TRUE if the device can disable the manual ejection of the media.

#### CAN SETVOLUME

Returns TRUE if the device supports software control of volume level.

#### COMPOUND DEVICE

Returns TRUE if the device requires an element name.

#### DEVICE TYPE

Returns one of the following:

- ampmix
- cdaudio
- cdxa
- digitalvideo
- overlay
- sequencer
- videodisc
- waveaudio
- other

#### HAS AUDIO

Returns TRUE if the device supports audio playback.

#### HAS VIDEO

Returns TRUE if the device supports video playback.

#### MESSAGE command

Returns TRUE if the device supports the command specified by **command**. Following are the commands you can query:

ACQUIRE	RECORD
CAPABILITY	RELEASE
CLOSE	RESUME
CONNECTION	SAVE
CONNECTOR	SEEK
CUE	SET
ESCAPE	SETCUEPOINT
GROUP	SETPOSITIONADVISE
INFO	SPIN
LOAD	STATUS
MASTERAUDIO	STEP
OPEN	STOP
PAUSE	SYSINFO
PLAY	

#### PREROLL TIME

Returns the deterministic or maximum notified preroll time in MMTIME units. A value of 0 for a notified preroll device indicates the preroll time is not bounded.

#### PREROLL TYPE

Returns the preroll characteristics of the device. Returns **notified** if the preroll time for the device is variable. Returns **deterministic** if the prerolled time for the device is fixed. Returns **none** if the device does not support preroll.

#### USES FILES

Returns TRUE if the device requires a file path name.

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.



---

# CAPABILITY (Required Command) - Syntax Diagram

CAPABILITY	object	CAN EJECT CAN PLAY CAN RECORD CAN SAVE CAN LOCKEJECT CAN SETVOLUME COMPOUND DEVICE DEVICE TYPE HAS AUDIO HAS VIDEO MESSAGE command PREROLL TIME PREROLL TYPE USES FILES	WAIT NOTIFY
------------	--------	--	----------------



---

# CAPABILITY (Required Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

# CLOSE

---

# CLOSE (Required Command) - Example

close digitalvideo wait

---

# CLOSE (Required Command) - Purpose

The CLOSE command closes the device context and frees resources.

---

## CLOSE (Required Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CLOSE (Required Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## CLOSE (Required Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CLOSE (Required Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# CLOSE (Required Command) - Syntax Diagram

CLOSE      object      WAIT  
NOTIFY

Examples

## CLOSE (Required Command) - Topics

- Select an item:
- Purpose
  - Syntax Diagram
  - Keywords
  - Example
  - Glossary

## INFO

## INFO (Required Command) - Example

info digitalvideo product

## INFO (Required Command) - Purpose

The INFO command fills a user-supplied buffer with information.

## INFO (Required Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## INFO (Required Command) Keyword - PRODUCT

**PRODUCT**

Returns a description of the hardware associated with a device. This usually includes the manufacturer and model information.

-----

## INFO (Required Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## INFO (Required Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## INFO (Required Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**PRODUCT**

Returns a description of the hardware associated with a device. This usually includes the manufacturer and model information.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

## INFO (Required Command) - Syntax Diagram

```
INFO          object          PRODUCT
                                     WAIT
                                     NOTIFY
```

## Examples

## INFO (Required Command) - Topics

Select an item:

- Purpose
- Syntax Diagram
- Keywords
- Example
- Glossary

OPEN

## OPEN (Required Command) - Example

```
open applause.wav shareable alias wfile wait
play wfile notify
```

## OPEN (Required Command) - Purpose

The OPEN command is used to open or create a new device instance.

OPEN returns a device ID that is used for subsequent calls for procedure interface, if desired.

---

## OPEN (Required Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## OPEN (Required Command) Keyword - ALIAS devicealias

### **ALIAS devicealias**

Specifies an alternate name for the given device. If an alias is specified, it must be used in subsequent references to avoid automatic open. Following are descriptions of what an alias can be:

- Any word that is not a keyword
- Any valid filename
- Any string of words enclosed in double quotes, for example:

"CD Player"

If a string is used, any leading and trailing blanks are ignored and internal blanks are preserved. Uppercase and lowercase can be used, but an alias is case insensitive.

---

## OPEN (Required Command) Keyword - DOSQUEUE

### **DOSQUEUE**

If a device instance is opened with the DOSQUEUE keyword specified, window handles that are passed in for the instance will be treated as OS/2 Control Program queue handles.

---

## OPEN (Required Command) Keyword - READONLY

### **READONLY**

Specifies that the file is to be opened in read-only mode.

---

## OPEN (Required Command) Keyword - SHAREABLE

**SHAREABLE**

Initializes the device as shareable. Specifying shareable makes the resources of the device available to other device contexts. If SHAREABLE is not specified with OPEN, the resource will be exclusively acquired when the device is opened.

---

## OPEN (Required Command) Keyword - TYPE devicetype

**TYPE devicetype**

Specifies the compound device used to control a device element. As an alternative to TYPE, an application can specify the name of a file to be opened. The media control interface uses the file EA or extension associated with the file to select the controlling device.

---

## OPEN (Required Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## OPEN (Required Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## OPEN (Required Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**ALIAS devicealias**

Specifies an alternate name for the given device. If an alias is specified, it must be used in subsequent references to avoid automatic open. Following are descriptions of what an alias can be:

- Any word that is not a keyword
- Any valid filename
- Any string of words enclosed in double quotes, for example:

"CD Player"

If a string is used, any leading and trailing blanks are ignored and internal blanks are preserved. Uppercase and lowercase can be used, but an alias is case insensitive.

**DOSQUEUE**

If a device instance is opened with the DOSQUEUE keyword specified, window handles that are passed in for the instance will be treated as OS/2 Control Program queue handles.

**READONLY**

Specifies that the file is to be opened in read-only mode.

**SHAREABLE**

Initializes the device as shareable. Specifying shareable makes the resources of the device available to other device contexts. If SHAREABLE is not specified with OPEN, the resource will be exclusively acquired when the device is opened.

**TYPE devicetype**

Specifies the compound device used to control a device element. As an alternative to TYPE, an application can specify the name of a file to be opened. The media control interface uses the file EA or extension associated with the file to select the controlling device.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

OPEN (Required Command) - Syntax Diagram

OPEN      object      ALIAS devicealias      WAIT  
                         DOSQUEUE      NOTIFY  
                         READONLY  
                         SHAREABLE  
                         TYPE devicetype



OPEN (Required Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

STATUS



---

## STATUS (Required Command) - Example

```
status waveaudio01 mode wait
```

---

## STATUS (Required Command) - Purpose

The STATUS command obtains status information for the device.

---

## STATUS (Required Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## STATUS (Required Command) Keyword - MODE

### MODE

Returns the current mode of the device: **not ready**, **stopped**, **playing**, **seeking**, **recording**, **paused**, or **other**.

---

## STATUS (Required Command) Keyword - READY

### READY

Returns TRUE if the device is ready.

---

## STATUS (Required Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

-----

# STATUS (Required Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

-----

# STATUS (Required Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**MODE**

Returns the current mode of the device: **not ready**, **stopped**, **playing**, **seeking**, **recording**, **paused**, or **other**.

**READY**

Returns TRUE if the device is ready.

**WAIT**

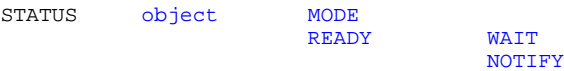
The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

-----

# STATUS (Required Command) - Syntax Diagram



---

## STATUS (Required Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## Basic Commands

In addition to the system and required commands, each device supports a set of device-type specific commands. Where possible, these type-specific commands are identical between device types. When type-specific commands are common to multiple devices, they are considered basic commands. For example, the basic [PLAY](#) command is identical for videodisc, wave audio, and CD audio players.

Although these commands are optional for a device, if a command is used it must recognize the options listed here and return MCIERR\_UNSUPPORTED\_FLAG for options that are not applicable.

For those devices that do not support a basic command, such as a [RECORD](#) command sent to a CD audio player, an MCIERR\_UNSUPPORTED\_FUNCTION is returned by that MCD. If a message is sent to a device that is not recognized, then MCIERR\_UNRECOGNIZED\_COMMAND is returned. Before using a basic command, an application can issue a [CAPABILITY](#) query to see if the device supports the command.

The following commands are basic commands:

- [CONNECTION](#)
  - [CONNECTOR](#)
  - [LOAD](#)
  - [PAUSE](#)
  - [PLAY](#)
  - [RECORD](#)
  - [RESUME](#)
  - [SAVE](#)
  - [SEEK](#)
  - [SET](#)
  - [SETCUEPOINT](#)
  - [SETPOSITIONADVISE](#)
  - [STATUS](#)
  - [STOP](#)
- 

## CONNECTOR

---

## CONNECTOR (Basic Command) - Example

```
connector waveaudio01 enable type microphone
```

---

## CONNECTOR (Basic Command) - Purpose

The CONNECTOR command enables, disables, or queries the status of connectors on a device.

---

## CONNECTOR (Basic Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CONNECTOR (Basic Command) Keyword - ENABLE

### **ENABLE**

Enables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both must also be specified.

---

## CONNECTOR (Basic Command) Keyword - DISABLE

### **DISABLE**

Disables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both must also be specified.

---

## CONNECTOR (Basic Command) Keyword - QUERY

### **QUERY**

Queries the status of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Use of this option requires that the NUMBER or TYPE keywords, or both must also be specified.

---

## CONNECTOR (Basic Command) Keyword - NUMBER

## connector\_number

### NUMBER connector\_number

The connector number on which to perform the requested action. If this item is omitted, the first connector is assumed. If the TYPE keyword is included, then the connector number is interpreted as a relative offset within the specified connector type.

-----

## CONNECTOR (Basic Command) Keyword - TYPE connector\_type

### TYPE connector\_type

The type of connector to which the requested action applies. See the TYPE keyword for [CONNECTION](#) for a list of connector types.

-----

## CONNECTOR (Basic Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## CONNECTOR (Basic Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CONNECTOR (Basic Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### ENABLE

Enables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both must also be specified.

### DISABLE

Disables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both must also be specified.

**QUERY**  
Queries the status of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Use of this option requires that the NUMBER or TYPE keywords, or both must also be specified.

**NUMBER connector\_number**  
The connector number on which to perform the requested action. If this item is omitted, the first connector is assumed. If the TYPE keyword is included, then the connector number is interpreted as a relative offset within the specified connector type.

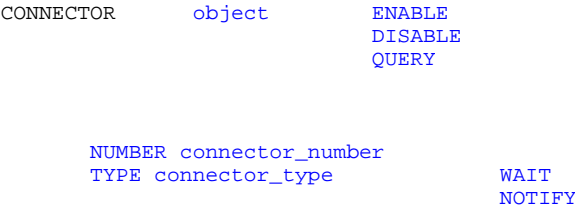
**TYPE connector\_type**  
The type of connector to which the requested action applies. See the TYPE keyword for [CONNECTION](#) for a list of connector types.

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CONNECTOR (Basic Command) - Syntax Diagram



---

## CONNECTOR (Basic Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

## LOAD

---

## LOAD (Basic Command) - Example

```
open digitalvideo01 alias video1 wait  
load video1 movie.avi wait
```

---

## LOAD (Basic Command) - Purpose

The LOAD command loads a new device element (file) into an already open device context.

---

## LOAD (Basic Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## LOAD (Basic Command) Keyword - filename

### **filename**

Name of the file to be loaded.

---

## LOAD (Basic Command) Keyword - NEW

### **NEW**

Creates a temporary element for subsequent use. The temporary file can be made permanent by providing a name using the [SAVE](#) command.

---

## LOAD (Basic Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

# LOAD (Basic Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# LOAD (Basic Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**filename**

Name of the file to be loaded.

**NEW**

Creates a temporary element for subsequent use. The temporary file can be made permanent by providing a name using the [SAVE](#) command.

**WAIT**

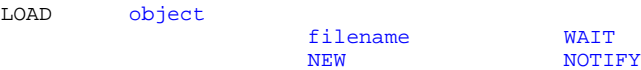
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# LOAD (Basic Command) - Syntax Diagram



-----

# LOAD (Basic Command) - Topics



Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

# PAUSE

---

## PAUSE (Basic Command) - Example

```
pause video1 wait
```

---

## PAUSE (Basic Command) - Purpose

The PAUSE command stops playing or recording. The difference between PAUSE and STOP is device dependent. On video devices, PAUSE generally continues to display the last frame, whereas STOP causes the display to blank. A device that is paused can frequently begin playing again with less latency than if it were stopped.

---

## PAUSE (Basic Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## PAUSE (Basic Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## PAUSE (Basic Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## PAUSE (Basic Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## PAUSE (Basic Command) - Syntax Diagram

PAUSE      [object](#)

[WAIT](#)  
[NOTIFY](#)

Examples

---

## PAUSE (Basic Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

# PLAY

---

## PLAY (Basic Command) - Example

```
play digitalvideo wait
```

---

## PLAY (Basic Command) - Purpose

The PLAY command starts playing the device.

**Note:** PLAY can be used on a file without a preceding OPEN.

---

## PLAY (Basic Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## PLAY (Basic Command) Keyword - FROM pos

### **FROM pos**

Specifies the position at which to start playing. If FROM is omitted, the device starts playing at the current position; if TO is omitted, the device **plays** to the end position.

---

## PLAY (Basic Command) Keyword - TO pos

**TO pos**

Specifies the position at which to stop playing. If FROM is omitted, the device starts playing at the current position; if TO is omitted, the device **plays** to the end position.

---

## PLAY (Basic Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## PLAY (Basic Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## PLAY (Basic Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**FROM pos**

Specifies the position at which to start playing. If FROM is omitted, the device starts playing at the current position; if TO is omitted, the device **plays** to the end position.

**TO pos**

Specifies the position at which to stop playing. If FROM is omitted, the device starts playing at the current position; if TO is omitted, the device **plays** to the end position.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## PLAY (Basic Command) - Syntax Diagram

PLAY      **object**

FROM pos      WAIT  
TO pos      NOTIFY

## Examples

# PLAY (Basic Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

# RECORD

# RECORD (Basic Command) - Example

```
record digitalvideo notify
```

# RECORD (Basic Command) - Purpose

The RECORD command starts recording data. By default, recording does not overwrite existing data but rather inserts data at the current position. On devices that cannot support inserting data (such as audio or video tape), recording overwrites existing data by default.

# RECORD (Basic Command) Keyword - object

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name

- Filename
- Alias

-----

## RECORD (Basic Command) Keyword - FROM pos

### FROM pos

Specifies the position at which to start recording. If FROM is omitted, the device starts recording at the current position; if TO is omitted, the device **records** to the end position.

-----

## RECORD (Basic Command) Keyword - TO pos

### TO pos

Specifies the position at which to stop recording. If FROM is omitted, the device starts recording at the current position; if TO is omitted, the device **records** to the end position.

-----

## RECORD (Basic Command) Keyword - INSERT

### INSERT

Data is to be added to the device element. This is the default on devices that support insertion of data (file-oriented devices). Returns MCI\_UNSUPPORTED\_FLAG on devices that do not support INSERT.

-----

## RECORD (Basic Command) Keyword - OVERWRITE

### OVERWRITE

Recorded data replaces existing data in the device element. This is the default on devices that do not support insertion of data (for example, videotape).

-----

## RECORD (Basic Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

# RECORD (Basic Command) Keyword - NOTIFY

**NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an `MM_MCINOTIFY` message is sent to the application window procedure.

## RECORD (Basic Command) - Keywords

**object**  
Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**FROM pos**  
Specifies the position at which to start recording. If FROM is omitted, the device starts recording at the current position; if TO is omitted, the device **records** to the end position.

**TO pos**  
Specifies the position at which to stop recording. If FROM is omitted, the device starts recording at the current position; if TO is omitted, the device **records** to the end position.

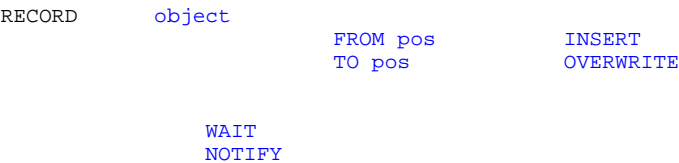
**INSERT**  
Data is to be added to the device element. This is the default on devices that support insertion of data (file-oriented devices). Returns `MCI_UNSUPPORTED_FLAG` on devices that do not support INSERT.

**OVERWRITE**  
Recorded data replaces existing data in the device element. This is the default on devices that do not support insertion of data (for example, videotape).

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an `MM_MCINOTIFY` message is sent to the application window procedure.

## RECORD (Basic Command) - Syntax Diagram



## Examples

---

# RECORD (Basic Command) - Topics

Select an item:

[Purpose](#)

[Syntax Diagram](#)

[Keywords](#)

[Example](#)

[Glossary](#)

---

# RESUME

---

## RESUME (Basic Command) - Example

```
resume waveaudio01 wait
```

---

## RESUME (Basic Command) - Purpose

The RESUME command resumes playing or recording from a paused state, keeping previously specified parameters in effect.

---

## RESUME (Basic Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

## RESUME (Basic Command) Keyword - WAIT



**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

# RESUME (Basic Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

-----

# RESUME (Basic Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**WAIT**

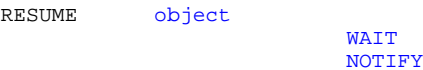
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

-----

# RESUME (Basic Command) - Syntax Diagram



-----

# RESUME (Basic Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

# SAVE

---

## SAVE (Basic Command) - Example

```
open macaw.avi alias videol wait  
save videol movie.avi wait
```

---

## SAVE (Basic Command) - Purpose

The SAVE command saves data for the device.

---

## SAVE (Basic Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## SAVE (Basic Command) Keyword - filename

### **filename**

The destination path and filename.

---

## SAVE (Basic Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## SAVE (Basic Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

-----

## SAVE (Basic Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**filename**

The destination path and filename.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

-----

## SAVE (Basic Command) - Syntax Diagram

```
SAVE          object      filename
                                     WAIT
                                     NOTIFY
```



-----

## SAVE (Basic Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## SEEK

---

### SEEK (Basic Command) - Example

```
seek digitalvideo to start wait
```

---

### SEEK (Basic Command) - Purpose

The SEEK command finds the specified position and stops.

---

### SEEK (Basic Command) Keyword - object

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

### SEEK (Basic Command) Keyword - TO pos

#### **TO pos**

The position at which to stop the seek. If it is greater than the length of the media, an OUT OF RANGE error is returned.

---

## SEEK (Basic Command) Keyword - TO START

### TO START

Seek to the beginning of the media.

-----

## SEEK (Basic Command) Keyword - TO END

### TO END

Seek to the end of the media.

-----

## SEEK (Basic Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## SEEK (Basic Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## SEEK (Basic Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### TO pos

The position at which to stop the seek. If it is greater than the length of the media, an OUT OF RANGE error is returned.

### TO START

Seek to the beginning of the media.

### TO END

Seek to the end of the media.

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## SEEK (Basic Command) - Syntax Diagram

```
SEEK      object      TO pos
                        TO START      WAIT
                        TO END        NOTIFY
```



---

## SEEK (Basic Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## SET

---

## SET (Basic Command) - Example

```
set waveaudio01 time format milliseconds wait
```

---

## SET (Basic Command) - Purpose

The SET command sets the various control items.

---

## SET (Basic Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## SET (Basic Command) Keyword - AUDIO

### **AUDIO**

Sets the audio attributes of the device context specified by the ALL, LEFT, RIGHT, OVER, and VOLUME keywords.

---

## SET (Basic Command) Keyword - ALL

### **ALL**

Apply to both or all of the channels (default).

Specify ON or OFF with the ALL keyword.

---

## SET (Basic Command) Keyword - ON

### **ON**

Enable audio output.

---

## SET (Basic Command) Keyword - OFF

### **OFF**

Disable audio output.

-----

# SET (Basic Command) Keyword - LEFT

**LEFT**  
Apply to the left channel.  
Specify ON or OFF with the LEFT keyword.

-----

# SET (Basic Command) Keyword - ON

**ON**  
Enable audio output to the left channel.

-----

# SET (Basic Command) Keyword - OFF

**OFF**  
Disable audio output to the left channel.

-----

# SET (Basic Command) Keyword - RIGHT

**RIGHT**  
Apply to the right channel.  
Specify ON or OFF with the RIGHT keyword.

-----

# SET (Basic Command) Keyword - ON

**ON**  
Enable audio output to the right channel.

-----

# SET (Basic Command) Keyword - OFF



**OFF**

Disable audio output to the right channel.

-----

## SET (Basic Command) Keyword - OVER milliseconds

**OVER milliseconds**

Apply the change over the specified time period (fade).

-----

## SET (Basic Command) Keyword - VOLUME percentage

**VOLUME percentage**

Set the device/mixer channel volume level.

-----

## SET (Basic Command) Keyword - DOOR CLOSED

**DOOR CLOSED**

Retracts the tray and closes the door, if possible.

-----

## SET (Basic Command) Keyword - DOOR OPEN

**DOOR OPEN**

Opens the door and ejects the tray, if possible.

-----

## SET (Basic Command) Keyword - DOOR LOCKED

**DOOR LOCKED**

Locks the media cover on the device (if any). This disables manual ejection of the media from the device.

-----

## SET (Basic Command) Keyword - DOOR UNLOCKED

#### **DOOR UNLOCKED**

Unlocks the media cover on the device (if any). This enables manual ejection of the media from the device.

---

## SET (Basic Command) Keyword - SPEED FORMAT PERCENTAGE

#### **SPEED FORMAT PERCENTAGE**

Sets the speed format to percentage.

---

## SET (Basic Command) Keyword - SPEED FORMAT FPS

#### **SPEED FORMAT FPS**

Sets the speed format to frames per second.

---

## SET (Basic Command) Keyword - TIME FORMAT MILLISECONDS

#### **TIME FORMAT MILLISECONDS**

Sets the time format, to milliseconds. You can abbreviate milliseconds as **ms**.

---

## SET (Basic Command) Keyword - TIME FORMAT MMTIME

#### **TIME FORMAT MMTIME**

Sets the time format to MMTIME.

---

## SET (Basic Command) Keyword - VIDEO OFF

#### **VIDEO OFF**

Disables video output.

---

## SET (Basic Command) Keyword - VIDEO ON

#### VIDEO ON

Enables video output.

---

## SET (Basic Command) Keyword - WAIT

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## SET (Basic Command) Keyword - NOTIFY

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## SET (Basic Command) - Keywords

#### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

#### AUDIO

Sets the audio attributes of the device context specified by the ALL, LEFT, RIGHT, OVER, and VOLUME keywords.

##### ALL

Apply to both or all of the channels (default).

Specify ON or OFF with the ALL keyword.

##### ON

Enable audio output.

##### OFF

Disable audio output.

##### LEFT

Apply to the left channel.

Specify ON or OFF with the LEFT keyword.

##### ON

Enable audio output to the left channel.

##### OFF

Disable audio output to the left channel.

**RIGHT**

Apply to the right channel.

Specify ON or OFF with the RIGHT keyword.

**ON**

Enable audio output to the right channel.

**OFF**

Disable audio output to the right channel.

**OVER milliseconds**

Apply the change over the specified time period (fade).

**VOLUME percentage**

Set the device/mixer channel volume level.

**DOOR CLOSED**

Retracts the tray and closes the door, if possible.

**DOOR OPEN**

Opens the door and ejects the tray, if possible.

**DOOR LOCKED**

Locks the media cover on the device (if any). This disables manual ejection of the media from the device.

**DOOR UNLOCKED**

Unlocks the media cover on the device (if any). This enables manual ejection of the media from the device.

**SPEED FORMAT PERCENTAGE**

Sets the speed format to percentage.

**SPEED FORMAT FPS**

Sets the speed format to frames per second.

**TIME FORMAT MILLISECONDS**

Sets the time format, to milliseconds. You can abbreviate milliseconds as **ms**.

**TIME FORMAT MMTIME**

Sets the time format to MMTIME.

**VIDEO OFF**

Disables video output.

**VIDEO ON**

Enables video output.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

-----

## SET (Basic Command) - Syntax Diagram

```
SET      object      AUDIO      ALL      ON
                                     OFF
                                     LEFT   ON
                                     OFF
                                     RIGHT  ON
```

```
                                OFF
                                OVER milliseconds
                                VOLUME percentage
DOOR CLOSED
DOOR OPEN
DOOR LOCKED
DOOR UNLOCKED
SPEED FORMAT PERCENTAGE
SPEED FORMAT FPS
TIME FORMAT MILLISECONDS
TIME FORMAT MMTIME
VIDEO OFF
VIDEO ON
```

```
WAIT
NOTIFY
```

## Examples

---

## SET (Basic Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## SETCUEPOINT

---

## SETCUEPOINT (Basic Command) - Example

```
setcuepoint waveaudio01 on at 5000 wait
```

---

## SETCUEPOINT (Basic Command) - Purpose

The SETCUEPOINT command sets a cue point. The window handle specified in the *hwndCallback* parameter of [mciSendString](#) receives the cue point notification ([MM\\_MCICUEPOINT](#)) messages.

This command is not related to the CUE command.

---

## SETCUEPOINT (Basic Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## SETCUEPOINT (Basic Command) Keyword - ON AT position

### **ON AT position**

Specifies the location of the cuepoint to enable in the currently set time format.

---

## SETCUEPOINT (Basic Command) Keyword - OFF AT position

### **OFF AT position**

Specifies the location of the cuepoint to disable in the currently set time format.

---

## SETCUEPOINT (Basic Command) Keyword - RETURN value

### **RETURN value**

A value to be returned in the user parameter field of the cue point notification message ([MM\\_MCICUEPOINT](#)).

---

## SETCUEPOINT (Basic Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## SETCUEPOINT (Basic Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## SETCUEPOINT (Basic Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**ON AT position**

Specifies the location of the cuepoint to enable in the currently set time format.

**OFF AT position**

Specifies the location of the cuepoint to disable in the currently set time format.

**RETURN value**

A value to be returned in the user parameter field of the cue point notification message ([MM\\_MCICUEPOINT](#)).

**WAIT**

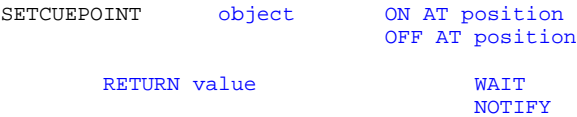
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## SETCUEPOINT (Basic Command) - Syntax Diagram



**Examples**

-----

## SETCUEPOINT (Basic Command) - Remarks

Devices that do not perform their own event detection might have less accurate cue points.

---

## SETCUEPOINT (Basic Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Remarks](#)  
[Example](#)  
[Glossary](#)

---

## SETPOSITIONADVISE

---

### SETPOSITIONADVISE (Basic Command) - Example

```
setpositionadvise waveaudio01 off wait
```

---

### SETPOSITIONADVISE (Basic Command) - Purpose

The SETPOSITIONADVISE command sets a position change notification for the device. The window handle specified in the *hwndCallback* parameter of [mciSendString](#) receives the position change notification ([MM\\_MCIPositionChange](#)) messages.

---

### SETPOSITIONADVISE (Basic Command) Keyword - object

#### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

### SETPOSITIONADVISE (Basic Command) Keyword - ON



**ON**

Enables the given position advise.

---

## SETPOSITIONADVISE (Basic Command) Keyword - EVERY units

**EVERY units**

The position change notification granularity in the currently set time format.

---

## SETPOSITIONADVISE (Basic Command) Keyword - OFF

**OFF**

Disables the given position advise.

---

## SETPOSITIONADVISE (Basic Command) Keyword - RETURN value

**RETURN value**

A value to be returned in the user parameter field of the position change notification message ([MM\\_MCIPOSITIONCHANGE](#)).

---

## SETPOSITIONADVISE (Basic Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## SETPOSITIONADVISE (Basic Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# SETPOSITIONADVISE (Basic Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

You must specify either the ON or OFF keyword.

## ON

Enables the given position advise.

## EVERY units

The position change notification granularity in the currently set time format.

## OFF

Disables the given position advise.

## RETURN value

A value to be returned in the user parameter field of the position change notification message ([MM\\_MCIPOSITIONCHANGE](#)).

## WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# SETPOSITIONADVISE (Basic Command) - Syntax Diagram

```
SETPOSITIONADVISE    object    ON    EVERY units
                     OFF
                     RETURN value    WAIT
                                     NOTIFY
```

Examples

-----

# SETPOSITIONADVISE (Basic Command) - Remarks

Devices that do not perform their own event detection might have less accurate position-advise events.

---

## SETPOSITIONADVISE (Basic Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Remarks](#)  
[Example](#)  
[Glossary](#)

---

## STATUS

---

### STATUS (Basic Command) - Example

```
status waveaudio01 volume wait
```

---

### STATUS (Basic Command) - Purpose

The STATUS command obtains status information for the device.

---

### STATUS (Basic Command) Keyword - object

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

### STATUS (Basic Command) Keyword - CURRENT TRACK

#### **CURRENT TRACK**

Returns the current track.

---

## STATUS (Basic Command) Keyword - LENGTH

#### **LENGTH**

Returns the total length of the segment. For compound devices, such as waveaudio, a device element must be opened or loaded to obtain the length.

---

## STATUS (Basic Command) Keyword - LENGTH TRACK number

#### **LENGTH TRACK number**

Returns the length of the track specified by **number**. Returned value is in MMTIME units unless the object has been opened and its time format has been changed.

---

## STATUS (Basic Command) Keyword - NUMBER OF TRACKS

#### **NUMBER OF TRACKS**

Returns the number of tracks on the media.

---

## STATUS (Basic Command) Keyword - POSITION

#### **POSITION**

Returns the current position.

---

## STATUS (Basic Command) Keyword - POSITION IN TRACK

#### **POSITION IN TRACK**

Returns the current position relative to the beginning of the track.

---

# STATUS (Basic Command) Keyword - POSITION TRACK number

**POSITION TRACK number**  
Returns the position of the start of the track specified by **number**. Returned value is in MMTIME units unless the object has been opened and its time format has been changed.

-----

# STATUS (Basic Command) Keyword - SPEED FORMAT

**SPEED FORMAT**  
Returns the speed format.

-----

# STATUS (Basic Command) Keyword - TIME FORMAT

**TIME FORMAT**  
Returns the time format.

-----

# STATUS (Basic Command) Keyword - VOLUME

**VOLUME**  
Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

-----

# STATUS (Basic Command) Keyword - WAIT

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

-----

# STATUS (Basic Command) Keyword - NOTIFY

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## STATUS (Basic Command) - Keywords

#### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

#### CURRENT TRACK

Returns the current track.

#### LENGTH

Returns the total length of the segment. For compound devices, such as waveaudio, a device element must be opened or loaded to obtain the length.

#### LENGTH TRACK number

Returns the length of the track specified by **number**. Returned value is in MMTIME units unless the object has been opened and its time format has been changed.

#### NUMBER OF TRACKS

Returns the number of tracks on the media.

#### POSITION

Returns the current position.

#### POSITION IN TRACK

Returns the current position relative to the beginning of the track.

#### POSITION TRACK number

Returns the position of the start of the track specified by **number**. Returned value is in MMTIME units unless the object has been opened and its time format has been changed.

#### SPEED FORMAT

Returns the speed format.

#### TIME FORMAT

Returns the time format.

#### VOLUME

Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## STATUS (Basic Command) - Syntax Diagram

STATUS	object	CURRENT TRACK	
		LENGTH	WAIT
		LENGTH TRACK number	NOTIFY
		NUMBER OF TRACKS	
		POSITION	
		POSITION IN TRACK	
		POSITION TRACK number	
		SPEED FORMAT	
		TIME FORMAT	
		VOLUME	

Examples

STATUS (Basic Command) - Topics

- Select an item:
- [Purpose](#)
- [Syntax Diagram](#)
- [Keywords](#)
- [Example](#)
- [Glossary](#)

STOP

STOP (Basic Command) - Example

```
stop digitalvideo01 wait
```

STOP (Basic Command) - Purpose

The STOP command stops the device.

STOP (Basic Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## STOP (Basic Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## STOP (Basic Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## STOP (Basic Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## STOP (Basic Command) - Syntax Diagram



STOP      object      WAIT  
NOTIFY

Examples

---

## STOP (Basic Command) - Topics

Select an item:  
[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## Audio Amplifier Mixer Commands

The audio amplifier mixer device supports extensions to the basic and required command sets. A device context of the audio amplifier mixer is a channel, either stereo or monaural, so most commands apply to channel levels. The exception is commands that apply to the final (output) mix, such as master volume.

Note that volume commands can be sent directly to player devices. These devices forward the volume command to the connected audio amplifier mixer channel device context when the output of the player is to an amplifier mixer. Other shaping functions, such as bass and treble, must be sent to the amplifier mixer.

The ampmix device is a conduit of information and relies on another device to provide the flow of information. Therefore, commands for the transport of information (such as play, seek, or stop), are sent to the attached device. Commands for transforming the information (such as treble or bass) are sent directly to the ampmix device. If the application needs to talk directly to the ampmix device, the value of the stream connector can be queried using the [CONNECTION](#) command, which returns a device context connection. An alias can be established for the connected device. Ampmix commands can then be sent directly to the ampmix device.

The ampmix device supports the device-type specific command, [MIXNOTIFY](#), and extensions to the following basic and required commands:

- [CAPABILITY](#)
- [CONNECTOR](#)
- [MIXNOTIFY](#)
- [SET](#)
- [STATUS](#)

---

## CAPABILITY

---

## CAPABILITY (Mixer Command) - Example

The following command returns FALSE.

```
capability ampmix01 can record wait
```

---

## CAPABILITY (Mixer Command) - Purpose

The CAPABILITY command requests additional information about the capabilities of the audio amplifier mixer device.

---

## CAPABILITY (Mixer Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

## CAPABILITY (Mixer Command) Keyword - CAN EJECT

### **CAN EJECT**

Returns FALSE.

---

## CAPABILITY (Mixer Command) Keyword - CAN LOCK EJECT

### **CAN LOCK EJECT**

Returns FALSE.

---

## CAPABILITY (Mixer Command) Keyword - CAN PLAY

### **CAN PLAY**

Returns FALSE.

---

## CAPABILITY (Mixer Command) Keyword - CAN PROCESS INTERNAL

**CAN PROCESS INTERNAL**  
Returns FALSE.

---

## CAPABILITY (Mixer Command) Keyword - CAN RECORD

**CAN RECORD**  
Returns FALSE.

---

## CAPABILITY (Mixer Command) Keyword - CAN SAVE

**CAN SAVE**  
Returns FALSE.

---

## CAPABILITY (Mixer Command) Keyword - CAN STREAM

**CAN STREAM**  
Returns FALSE.

---

## CAPABILITY (Mixer Command) Keyword - CAN SET VOLUME

**CAN SET VOLUME**  
Returns TRUE.

---

## CAPABILITY (Mixer Command) Keyword - COMPOUND DEVICE

**COMPOUND DEVICE**  
Returns FALSE.

---

## CAPABILITY (Mixer Command) Keyword - DEVICE TYPE

**DEVICE TYPE**  
Returns **Ampmix**.

---

## CAPABILITY (Mixer Command) Keyword - HAS AUDIO

**HAS AUDIO**  
Returns TRUE.

---

## CAPABILITY (Mixer Command) Keyword - HAS VIDEO

**HAS VIDEO**  
Returns FALSE.

---

## CAPABILITY (Mixer Command) Keyword - MESSAGE command

**MESSAGE command**  
Returns TRUE if the device supports the command specified by **command** The **command** can be any of the string commands such as OPEN, PLAY, and so on.

---

## CAPABILITY (Mixer Command) Keyword - PREROLL TIME

**PREROLL TIME**  
Returns 0, indicating the preroll time is not bounded.

---

# CAPABILITY (Mixer Command) Keyword - PREROLL TYPE

**PREROLL TYPE**  
Returns NONE.

-----

# CAPABILITY (Mixer Command) Keyword - USES FILES

**USES FILES**  
Returns FALSE.

-----

# CAPABILITY (Mixer Command) Keyword - EXTENDED MIXER CONNECTOR type

**EXTENDED MIXER CONNECTOR type**  
Indicates that a mixer format will be queried. The **type** specified must be a valid connector type. If the EXTENDED MIXER keywords are specified, the SUPPORTS keyword must also be specified.

-----

# CAPABILITY (Mixer Command) Keyword - SUPPORTS AUTO LEVEL CONTROL

**SUPPORTS AUTO LEVEL CONTROL**  
Indicates whether auto-level control settings are supported by the connector specified.

-----

# CAPABILITY (Mixer Command) Keyword - SUPPORTS BALANCE

**SUPPORTS BALANCE**  
Indicates whether balance settings are supported by the connector specified.

-----

# CAPABILITY (Mixer Command) Keyword - SUPPORTS BASS

**SUPPORTS BASS**

Indicates whether bass settings are supported by the connector specified.

-----

**CAPABILITY (Mixer Command) Keyword - SUPPORTS CHORUS**

**SUPPORTS CHORUS**

Indicates whether chorus settings are supported by the connector specified.

-----

**CAPABILITY (Mixer Command) Keyword - SUPPORTS CROSSOVER**

**SUPPORTS CROSSOVER**

Indicates whether crossover settings are supported by the connector specified.

-----

**CAPABILITY (Mixer Command) Keyword - SUPPORTS CUSTOM1**

**SUPPORTS CUSTOM1**

Indicates whether a custom effect is supported by the connector specified.

-----

**CAPABILITY (Mixer Command) Keyword - SUPPORTS CUSTOM2**

**SUPPORTS CUSTOM2**

Indicates whether a custom effect is supported by the connector specified.

-----

**CAPABILITY (Mixer Command) Keyword - SUPPORTS CUSTOM3**

**SUPPORTS CUSTOM3**

Indicates whether a custom effect is supported by the connector specified.

---

## CAPABILITY (Mixer Command) Keyword - SUPPORTS GAIN

**SUPPORTS GAIN**

Indicates whether gain settings are supported by the connector specified.

---

## CAPABILITY (Mixer Command) Keyword - SUPPORTS LOUDNESS

**SUPPORTS LOUDNESS**

Indicates whether loudness settings are supported by the connector specified.

---

## CAPABILITY (Mixer Command) Keyword - SUPPORTS MID

**SUPPORTS MID**

Indicates whether mid settings are supported by the connector specified.

---

## CAPABILITY (Mixer Command) Keyword - SUPPORTS MONITOR

**SUPPORTS MONITOR**

Indicates whether monitor settings are supported by the connector specified.

---

## CAPABILITY (Mixer Command) Keyword - SUPPORTS MUTE

**SUPPORTS MUTE**

Indicates whether mute settings are supported by the connector specified.

-----

# CAPABILITY (Mixer Command) Keyword - SUPPORTS PITCH

**SUPPORTS PITCH**  
Indicates whether pitch settings are supported by the connector specified.

-----

# CAPABILITY (Mixer Command) Keyword - SUPPORTS REVERB

**SUPPORTS REVERB**  
Indicates whether reverb settings are supported by the connector specified.

-----

# CAPABILITY (Mixer Command) Keyword - SUPPORTS STEREO ENHANCE

**SUPPORTS STEREO ENHANCE**  
Indicates whether stereo enhance settings are supported by the connector specified.

-----

# CAPABILITY (Mixer Command) Keyword - SUPPORTS TREBLE

**SUPPORTS TREBLE**  
Indicates whether treble settings are supported by the connector specified.

-----

# CAPABILITY (Mixer Command) Keyword - SUPPORTS VOLUME

**SUPPORTS VOLUME**  
Indicates whether volume settings are supported by the connector specified.



---

## CAPABILITY (Mixer Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

---

## CAPABILITY (Mixer Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CAPABILITY (Mixer Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### **CAN EJECT**

Returns FALSE.

### **CAN LOCK EJECT**

Returns FALSE.

### **CAN PLAY**

Returns FALSE.

### **CAN PROCESS INTERNAL**

Returns FALSE.

### **CAN RECORD**

Returns FALSE.

### **CAN SAVE**

Returns FALSE.

### **CAN STREAM**

Returns FALSE.

### **CAN SET VOLUME**

Returns TRUE.

### **COMPOUND DEVICE**

Returns FALSE.

### **DEVICE TYPE**

Returns **Ampmix**.

#### **HAS AUDIO**

Returns TRUE.

#### **HAS VIDEO**

Returns FALSE.

#### **MESSAGE command**

Returns TRUE if the device supports the command specified by **command**. The **command** can be any of the string commands such as OPEN, PLAY, and so on.

#### **PREROLL TIME**

Returns 0, indicating the preroll time is not bounded.

#### **PREROLL TYPE**

Returns NONE.

#### **USES FILES**

Returns FALSE.

#### **EXTENDED MIXER CONNECTOR type**

Indicates that a mixer format will be queried. The **type** specified must be a valid connector type. If the EXTENDED MIXER keywords are specified, the SUPPORTS keyword must also be specified.

##### **SUPPORTS AUTO LEVEL CONTROL**

Indicates whether auto-level control settings are supported by the connector specified.

##### **SUPPORTS BALANCE**

Indicates whether balance settings are supported by the connector specified.

##### **SUPPORTS BASS**

Indicates whether bass settings are supported by the connector specified.

##### **SUPPORTS CHORUS**

Indicates whether chorus settings are supported by the connector specified.

##### **SUPPORTS CROSSOVER**

Indicates whether crossover settings are supported by the connector specified.

##### **SUPPORTS CUSTOM1**

Indicates whether a custom effect is supported by the connector specified.

##### **SUPPORTS CUSTOM2**

Indicates whether a custom effect is supported by the connector specified.

##### **SUPPORTS CUSTOM3**

Indicates whether a custom effect is supported by the connector specified.

##### **SUPPORTS GAIN**

Indicates whether gain settings are supported by the connector specified.

##### **SUPPORTS LOUDNESS**

Indicates whether loudness settings are supported by the connector specified.

##### **SUPPORTS MID**

Indicates whether mid settings are supported by the connector specified.

##### **SUPPORTS MONITOR**

Indicates whether monitor settings are supported by the connector specified.

##### **SUPPORTS MUTE**

Indicates whether mute settings are supported by the connector specified.

##### **SUPPORTS PITCH**

Indicates whether pitch settings are supported by the connector specified.

##### **SUPPORTS REVERB**

Indicates whether reverb settings are supported by the connector specified.

##### **SUPPORTS STEREO ENHANCE**

Indicates whether stereo enhance settings are supported by the connector specified.

**SUPPORTS TREBLE**  
Indicates whether treble settings are supported by the connector specified.

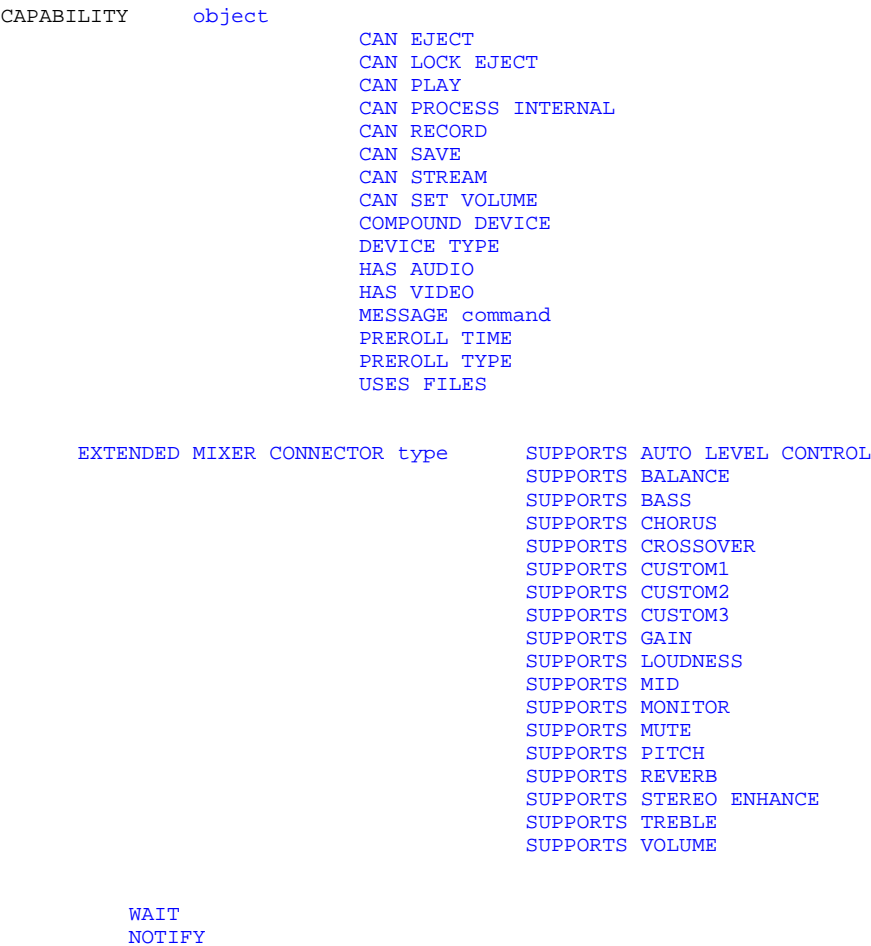
**SUPPORTS VOLUME**  
Indicates whether volume settings are supported by the connector specified.

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

**NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

-----

# CAPABILITY (Mixer Command) - Syntax Diagram



Examples

-----

# CAPABILITY (Mixer Command) - Topics

Select an item:

[Purpose](#)

[Syntax Diagram](#)

[Keywords](#)

[Example](#)

[Glossary](#)

---

## CONNECTOR

---

## CONNECTOR (Mixer Command) - Example

The following command returns TRUE.

```
connector wave query type amp stream wait
```

---

## CONNECTOR (Mixer Command) - Purpose

The CONNECTOR command enables, disables, or queries the status of connectors on a device.

---

## CONNECTOR (Mixer Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CONNECTOR (Mixer Command) Keyword - ENABLE

### **ENABLE**

Enables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both

also be specified.

-----

## CONNECTOR (Mixer Command) Keyword - DISABLE

### **DISABLE**

Disables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

-----

## CONNECTOR (Mixer Command) Keyword - QUERY

### **QUERY**

Queries the status of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

-----

## CONNECTOR (Mixer Command) Keyword - NUMBER connector\_number

### **NUMBER connector\_number**

Indicates the connector number on which to perform the requested action. If this item is omitted, then the first connector is assumed. If the TYPE keyword is included, then the connector number is interpreted as a relative offset within the specified connector type.

-----

## CONNECTOR (Mixer Command) Keyword - TYPE connector\_type

### **TYPE connector\_type**

Indicates the type of connector to which the requested action applies. The following connector types are supported by this device.

amp stream

Digital input or output for the audio amplifier/mixer. This connector is always enabled.

line in

The line input connector. This connector is usually attached to the line out connector of another device such as a tape player or other audio input source.

microphone

The microphone connector. This connector is usually attached to a microphone for live recording or voice annotation.

line out

The line output connector. This connector is usually attached to the line in connector of another device such as a tape recorder or other audio device.

speakers	The speakers connector. This connector is usually attached to a pair of external or internal speakers.
headphones	The headphones connector. This connector is usually attached to a pair of headphones.

-----

## CONNECTOR (Mixer Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## CONNECTOR (Mixer Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CONNECTOR (Mixer Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### ENABLE

Enables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

### DISABLE

Disables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

### QUERY

Queries the status of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

### NUMBER connector\_number

Indicates the connector number on which to perform the requested action. If this item is omitted, then the first connector is assumed. If the TYPE keyword is included, then the connector number is interpreted as a relative offset within the specified connector type.

### TYPE connector\_type

Indicates the type of connector to which the requested action applies. The following connector types are supported by this device.

amp stream	Digital input or output for the audio amplifier/mixer. This connector is always enabled.
------------	--

line in	The line input connector. This connector is usually attached to the line out connector of another device such as a tape player or other audio input source.
microphone	The microphone connector. This connector is usually attached to a microphone for live recording or voice annotation.
line out	The line output connector. This connector is usually attached to the line in connector of another device such as a tape recorder or other audio device.
speakers	The speakers connector. This connector is usually attached to a pair of external or internal speakers.
headphones	The headphones connector. This connector is usually attached to a pair of headphones.
<b>WAIT</b>	
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.	
<b>NOTIFY</b>	
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an <a href="#">MM_MCINOTIFY</a> message is sent to the application window procedure.	

## CONNECTOR (Mixer Command) - Syntax Diagram

```

CONNECTOR      object      ENABLE
                  DISABLE
                  QUERY

NUMBER connector_number
TYPE connector_type

WAIT
NOTIFY

```



## CONNECTOR (Mixer Command) - Topics

Select an item:

- [Purpose](#)
- [Syntax Diagram](#)
- [Keywords](#)
- [Example](#)
- [Glossary](#)

## MIXNOTIFY

---

## MIXNOTIFY (Mixer Command) - Example

```
mixnotify ampmix01 on
```

---

## MIXNOTIFY (Mixer Command) - Purpose

The MIXNOTIFY command notifies an application when a mixer attribute (such as treble, bass, and so on) changes.

---

## MIXNOTIFY (Mixer Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## MIXNOTIFY (Mixer Command) Keyword - ON

### **ON**

Turns mixer notifications on.

---

## MIXNOTIFY (Mixer Command) Keyword - OFF

### **OFF**

Turns mixer notifications off.

---

## MIXNOTIFY (Mixer Command) Keyword - WAIT



**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

-----

# MIXNOTIFY (Mixer Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

-----

# MIXNOTIFY (Mixer Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**ON**

Turns mixer notifications on.

**OFF**

Turns mixer notifications off.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

-----

# MIXNOTIFY (Mixer Command) - Syntax Diagram

MIXNOTIFY      object      ON  
                                 OFF      WAIT  
   NOTIFY



---

## MIXNOTIFY (Mixer Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## SET

---

## SET (Mixer Command) - Example

```
set ampmix01 audio gain 80 wait
```

---

## SET (Mixer Command) - Purpose

The SET command sets various control items for the audio amplifier mixer.

---

## SET (Mixer Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## SET (Mixer Command) Keyword - AUDIO

### **AUDIO**

The audio attributes of the device context specified by the ALL, LEFT, RIGHT, OVER, and VOLUME keywords.

---

## SET (Mixer Command) Keyword - ALL

### ALL

Applies to both or all channels (default).

Specify ON or OFF with the ALL keyword.

---

## SET (Mixer Command) Keyword - ON

### ON

Enable audio output.

---

## SET (Mixer Command) Keyword - OFF

### OFF

Disable audio output.

---

## SET (Mixer Command) Keyword - LEFT

### LEFT

Applies to left channel.

Specify ON or OFF with the LEFT keyword.

---

## SET (Mixer Command) Keyword - ON

### ON

Enable audio output to the left audio channel.

---

## SET (Mixer Command) Keyword - OFF

**OFF**

Disable audio output to the left audio channel.

-----

## SET (Mixer Command) Keyword - RIGHT

**RIGHT**

Applies to right channel.

Specify ON or OFF with the RIGHT keyword.

-----

## SET (Mixer Command) Keyword - ON

**ON**

Enable audio output to the right audio channel.

-----

## SET (Mixer Command) Keyword - OFF

**OFF**

Disable audio output to the right audio channel.

-----

## SET (Mixer Command) Keyword - BASS level

**BASS level**

Sets the bass level as a percentage of the maximum achievable effect. The effect applies to the final output mix. Any specification of a channel is ignored.

-----

## SET (Mixer Command) Keyword - TREBLE level

**TREBLE level**

Sets the mixer-channel treble level as a percentage of the maximum achievable effect. The effect applies to the final output mix. Any specification of a channel is ignored.

---

## SET (Mixer Command) Keyword - BALANCE level

### **BALANCE level**

Sets the balance level. Zero is defined as full left balance while one hundred is defined as full right balance. This value is ignored for monaural channels. This effect applies to the final output mix. Any specification of a channel is ignored.

---

## SET (Mixer Command) Keyword - PITCH level

### **PITCH level**

Sets the pitch as a percentage of the maximum achievable effect. This effect applies to the final output mix. Any specification of a channel is ignored.

---

## SET (Mixer Command) Keyword - GAIN level

### **GAIN level**

Sets the gain as a percentage of the maximum achievable effect for the currently selected input.

---

## SET (Mixer Command) Keyword - MONITOR

### **MONITOR**

Sets the amplifier mixer device to monitor, or not to monitor, the input signal from one device while the output of another device is being recorded. This option should be used along with the ON and OFF keywords.

Specify ON or OFF with the MONITOR keyword.

---

## SET (Mixer Command) Keyword - ON

### **ON**

Enables monitoring of the input signal.

---

## SET (Mixer Command) Keyword - OFF

**OFF**

Disables monitoring of the input signal.

---

## SET (Mixer Command) Keyword - OVER milliseconds

**OVER milliseconds**

Apply the change over the specified time period (fade).

---

## SET (Mixer Command) Keyword - VOLUME level

**VOLUME level**

Sets the mixer-channel volume level as a percentage of the maximum achievable effect. The precise channel is specified by using the ALL, LEFT, or RIGHT keywords.

---

## SET (Mixer Command) Keyword - CONNECTOR type

**CONNECTOR type**

Specifies one of the following connector types for which the settings are to apply.

- amp stream
  - audio in
  - audio out
  - headphones
  - internalaudio
  - microphone
  - midi in
  - midi out
  - midi stream
  - line in
  - line out
  - null
  - phone set
  - phone line
  - speakers
  - universal
- 

## SET (Mixer Command) Keyword - MIXER AUTO LEVEL CONTROL level

**MIXER AUTO LEVEL CONTROL level**

Sets the auto-level control setting for the specified connector as a percentage (0-100).

-----

## SET (Mixer Command) Keyword - MIXER BALANCE level

### **MIXER BALANCE level**

Sets the balance setting for the specified connector as a percentage (0-100).

-----

## SET (Mixer Command) Keyword - MIXER BASS level

### **MIXER BASS level**

Sets the bass setting for the specified connector as a percentage (0-100).

-----

## SET (Mixer Command) Keyword - MIXER CHORUS level

### **MIXER CHORUS level**

Sets the chorus setting for the specified connector as a percentage (0-100).

-----

## SET (Mixer Command) Keyword - MIXER CROSSOVER level

### **MIXER CROSSOVER level**

Sets the crossover setting for the specified connector as a percentage (0-100).

-----

## SET (Mixer Command) Keyword - MIXER CUSTOM1 level

### **MIXER CUSTOM1 level**

Sets the custom effect setting for the specified connector as a percentage (0-100).

-----

## SET (Mixer Command) Keyword - MIXER CUSTOM2 level

**MIXER CUSTOM2 level**

Sets the custom effect setting for the specified connector as a percentage (0-100).

-----

SET (Mixer Command) Keyword - MIXER CUSTOM3 level

**MIXER CUSTOM3 level**

Sets the custom effect setting for the specified connector as a percentage (0-100).

-----

SET (Mixer Command) Keyword - MIXER GAIN level

**MIXER GAIN level**

Sets the gain setting for the specified connector as a percentage (0-100).

-----

SET (Mixer Command) Keyword - MIXER LOUDNESS level

**MIXER LOUDNESS level**

Sets the loudness setting for the specified connector as a percentage (0-100).

-----

SET (Mixer Command) Keyword - MIXER MID level

**MIXER MID level**

Sets the mid setting for the specified connector as a percentage (0-100).

-----

SET (Mixer Command) Keyword - MIXER MONITOR level

**MIXER MONITOR level**

Sets the monitor setting for the specified connector as a percentage (0-100).

-----

SET (Mixer Command) Keyword - MIXER MUTE



#### **MIXER MUTE**

Sets the specified connector to mute.

-----

## SET (Mixer Command) Keyword - MIXER PITCH level

#### **MIXER PITCH level**

Sets the pitch setting for the specified connector as a percentage (0-100).

-----

## SET (Mixer Command) Keyword - MIXER REVERB level

#### **MIXER REVERB level**

Sets the reverb setting for the specified connector as a percentage (0-100).

-----

## SET (Mixer Command) Keyword - MIXER STEREO ENHANCE level

#### **MIXER STEREO ENHANCE level**

Sets the stereo enhancement setting for the specified connector as a percentage (0-100).

-----

## SET (Mixer Command) Keyword - MIXER TREBLE level

#### **MIXER TREBLE level**

Sets the treble setting for the specified connector as a percentage (0-100).

-----

## SET (Mixer Command) Keyword - MIXER VOLUME level

#### **MIXER VOLUME level**

Sets the volume setting for the specified connector as a percentage (0-100).

-----

## SET (Mixer Command) Keyword - WAIT

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## SET (Mixer Command) Keyword - NOTIFY

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## SET (Mixer Command) - Keywords

#### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

#### AUDIO

The audio attributes of the device context specified by the ALL, LEFT, RIGHT, OVER, and VOLUME keywords.

##### ALL

Applies to both or all channels (default).

Specify ON or OFF with the ALL keyword.

##### ON

Enable audio output.

##### OFF

Disable audio output.

##### LEFT

Applies to left channel.

Specify ON or OFF with the LEFT keyword.

##### ON

Enable audio output to the left audio channel.

##### OFF

Disable audio output to the left audio channel.

##### RIGHT

Applies to right channel.

Specify ON or OFF with the RIGHT keyword.

##### ON

Enable audio output to the right audio channel.

##### OFF

Disable audio output to the right audio channel.

**BASS level**

Sets the bass level as a percentage of the maximum achievable effect. The effect applies to the final output mix. Any specification of a channel is ignored.

**TREBLE level**

Sets the mixer-channel treble level as a percentage of the maximum achievable effect. The effect applies to the final output mix. Any specification of a channel is ignored.

**BALANCE level**

Sets the balance level. Zero is defined as full left balance while one hundred is defined as full right balance. This value is ignored for monaural channels. This effect applies to the final output mix. Any specification of a channel is ignored.

**PITCH level**

Sets the pitch as a percentage of the maximum achievable effect. This effect applies to the final output mix. Any specification of a channel is ignored.

**GAIN level**

Sets the gain as a percentage of the maximum achievable effect for the currently selected input.

**MONITOR**

Sets the amplifier mixer device to monitor, or not to monitor, the input signal from one device while the output of another device is being recorded. This option should be used along with the ON and OFF keywords.

Specify ON or OFF with the MONITOR keyword.

**ON**

Enables monitoring of the input signal.

**OFF**

Disables monitoring of the input signal.

**OVER milliseconds**

Apply the change over the specified time period (fade).

**VOLUME level**

Sets the mixer-channel volume level as a percentage of the maximum achievable effect. The precise channel is specified by using the ALL, LEFT, or RIGHT keywords.

**CONNECTOR type**

Specifies one of the following connector types for which the settings are to apply.

- amp stream
- audio in
- audio out
- headphones
- internalaudio
- microphone
- midi in
- midi out
- midi stream
- line in
- line out
- null
- phone set
- phone line
- speakers
- universal

**MIXER AUTO LEVEL CONTROL level**

Sets the auto-level control setting for the specified connector as a percentage (0-100).

**MIXER BALANCE level**

Sets the balance setting for the specified connector as a percentage (0-100).

**MIXER BASS level**

Sets the bass setting for the specified connector as a percentage (0-100).

**MIXER CHORUS level**

Sets the chorus setting for the specified connector as a percentage (0-100).

**MIXER CROSSOVER level**

Sets the crossover setting for the specified connector as a percentage (0-100).

**MIXER CUSTOM1 level**

Sets the custom effect setting for the specified connector as a percentage (0-100).

**MIXER CUSTOM2 level**

Sets the custom effect setting for the specified connector as a percentage (0-100).

**MIXER CUSTOM3 level**

Sets the custom effect setting for the specified connector as a percentage (0-100).

**MIXER GAIN level**

Sets the gain setting for the specified connector as a percentage (0-100).

**MIXER LOUDNESS level**

Sets the loudness setting for the specified connector as a percentage (0-100).

**MIXER MID level**

Sets the mid setting for the specified connector as a percentage (0-100).

**MIXER MONITOR level**

Sets the monitor setting for the specified connector as a percentage (0-100).

**MIXER MUTE**

Sets the specified connector to mute.

**MIXER PITCH level**

Sets the pitch setting for the specified connector as a percentage (0-100).

**MIXER REVERB level**

Sets the reverb setting for the specified connector as a percentage (0-100).

**MIXER STEREO ENHANCE level**

Sets the stereo enhancement setting for the specified connector as a percentage (0-100).

**MIXER TREBLE level**

Sets the treble setting for the specified connector as a percentage (0-100).

**MIXER VOLUME level**

Sets the volume setting for the specified connector as a percentage (0-100).

**WAIT**

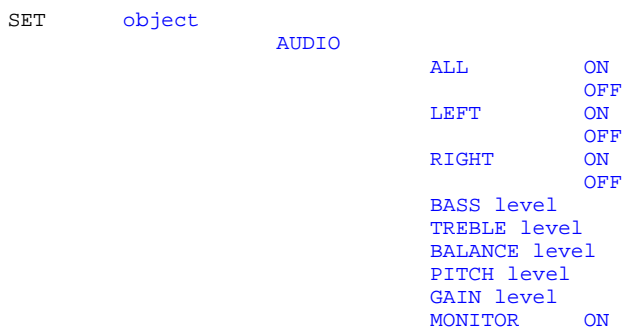
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## SET (Mixer Command) - Syntax Diagram



	OFF	
	OVER milliseconds	
	VOLUME level	
CONNECTOR type	MIXER AUTO LEVEL CONTROL level	WAIT
	MIXER BASS level	NOTIFY
	MIXER BALANCE level	
	MIXER CHORUS level	
	MIXER CROSSOVER level	
	MIXER CUSTOM1 level	
	MIXER CUSTOM2 level	
	MIXER CUSTOM3 level	
	MIXER GAIN level	
	MIXER LOUDNESS level	
	MIXER MID level	
	MIXER MONITOR level	
	MIXER MUTE	
	MIXER PITCH level	
	MIXER REVERB level	
	MIXER STEREO ENHANCE level	
	MIXER TREBLE level	
	MIXER VOLUME level	

Examples

## SET (Mixer Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

## STATUS

## STATUS (Mixer Command) - Example

```
status ampmix01 balance wait
```

## STATUS (Mixer Command) - Purpose

The STATUS command obtains status information for the device.

---

## STATUS (Mixer Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## STATUS (Mixer Command) Keyword - AUDIO CHANNEL value

### AUDIO CHANNEL value

Must specify **all**, **left**, **right**, or a *number*. Returns TRUE or FALSE to indicate whether the channel is enabled or disabled, respectively.

---

## STATUS (Mixer Command) Keyword - BASS

### BASS

Returns the current bass setting as a percentage of the maximum achievable effect.

---

## STATUS (Mixer Command) Keyword - BALANCE

### BALANCE

Returns the current balance setting where zero is defined as full left balance and one hundred is defined as full right balance.

---

## STATUS (Mixer Command) Keyword - CURRENT TRACK

### CURRENT TRACK

Returns the current function. The amplifier mixer device always returns 1.

---

## STATUS (Mixer Command) Keyword - GAIN

#### **GAIN**

Returns the current gain setting as a percentage of the maximum achievable effect.

-----

## STATUS (Mixer Command) Keyword - LENGTH

#### **LENGTH**

Returns the total length of the segment. The amplifier mixer device does not support this function.

-----

## STATUS (Mixer Command) Keyword - LENGTH TRACK number

#### **LENGTH TRACK number**

Returns the length of the track specified by **number**. The amplifier mixer device does not support this function.

-----

## STATUS (Mixer Command) Keyword - MONITOR

#### **MONITOR**

Returns whether or not the amplifier mixer device is monitoring the input signal from one device while the output of another device is being recorded. Either TRUE or FALSE will be returned.

-----

## STATUS (Mixer Command) Keyword - NUMBER OF TRACKS

#### **NUMBER OF TRACKS**

Returns the number of tracks on the media. The amplifier mixer device does not support this function.

-----

## STATUS (Mixer Command) Keyword - PITCH

#### **PITCH**

Returns the current pitch setting as a percentage of the maximum achievable effect.

---

## STATUS (Mixer Command) Keyword - POSITION

### POSITION

Returns the current position. The amplifier mixer device does not support this function.

---

## STATUS (Mixer Command) Keyword - POSITION IN TRACK

### POSITION IN TRACK

Returns the current position relative to the beginning of the track. The amplifier mixer device does not support this function.

---

## STATUS (Mixer Command) Keyword - POSITION TRACK number

### POSITION TRACK number

Returns the position of the start of the track specified by **number**. The amplifier mixer device does not support this function.

---

## STATUS (Mixer Command) Keyword - SPEED FORMAT

### SPEED FORMAT

Returns the current speed format. The amplifier mixer device does not support this function.

---

## STATUS (Mixer Command) Keyword - TIME FORMAT

### TIME FORMAT

Returns the current time format. The amplifier mixer device does not support this function.

---

## STATUS (Mixer Command) Keyword - TREBLE

### TREBLE



Returns the current treble setting as a percentage of the maximum achievable effect.

-----

## STATUS (Mixer Command) Keyword - VOLUME

### **VOLUME**

Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

-----

## STATUS (Mixer Command) Keyword - CONNECTOR type

### **CONNECTOR type**

Specifies the connector that status information is desired from. The connector specified must be a valid connector.

-----

## STATUS (Mixer Command) Keyword - AUTO LEVEL CONTROL

### **AUTO LEVEL CONTROL**

Returns the auto-level control setting for the desired connector.

-----

## STATUS (Mixer Command) Keyword - BALANCE

### **BALANCE**

Returns the balance setting for the desired connector.

-----

## STATUS (Mixer Command) Keyword - BASS

### **BASS**

Returns the bass setting for the desired connector.

-----

## STATUS (Mixer Command) Keyword - CHORUS

**CHORUS**  
Returns the chorus setting for the desired connector.

-----

STATUS (Mixer Command) Keyword - CROSSOVER

**CROSSOVER**  
Returns the crossover setting for the desired connector.

-----

STATUS (Mixer Command) Keyword - CUSTOM1

**CUSTOM1**  
Returns the custom effect setting for the desired connector.

-----

STATUS (Mixer Command) Keyword - CUSTOM2

**CUSTOM2**  
Returns the custom effect setting for the desired connector.

-----

STATUS (Mixer Command) Keyword - CUSTOM3

**CUSTOM3**  
Returns the custom effect setting for the desired connector.

-----

STATUS (Mixer Command) Keyword - GAIN

**GAIN**  
Returns the gain setting for the desired connector.

-----

STATUS (Mixer Command) Keyword - LOUDNESS

**LOUDNESS**

Returns the loudness setting for the desired connector.

-----

**STATUS (Mixer Command) Keyword - MID**

**MID**

Returns the mid setting for the desired connector.

-----

**STATUS (Mixer Command) Keyword - MONITOR**

**MONITOR**

Returns the monitor setting for the desired connector.

-----

**STATUS (Mixer Command) Keyword - MUTE**

**MUTE**

Returns the mute setting for the desired connector.

-----

**STATUS (Mixer Command) Keyword - PITCH**

**PITCH**

Returns the pitch setting for the desired connector.

-----

**STATUS (Mixer Command) Keyword - REVERB**

**REVERB**

Returns the reverb setting for the desired connector.

-----

**STATUS (Mixer Command) Keyword - TREBLE**

#### TREBLE

Returns the treble setting for the desired connector.

---

## STATUS (Mixer Command) Keyword - VOLUME

#### VOLUME

Returns the volume setting for the desired connector.

---

## STATUS (Mixer Command) Keyword - WAIT

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified to receive return string information.

---

## STATUS (Mixer Command) Keyword - NOTIFY

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

---

## STATUS (Mixer Command) - Keywords

#### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

#### AUDIO CHANNEL value

Must specify **all**, **left**, **right**, or a *number*. Returns TRUE or FALSE to indicate whether the channel is enabled or disabled, respectively.

#### BASS

Returns the current bass setting as a percentage of the maximum achievable effect.

#### BALANCE

Returns the current balance setting where zero is defined as full left balance and one hundred is defined as full right balance.

#### CURRENT TRACK

Returns the current function. The amplifier mixer device always returns 1.

#### **GAIN**

Returns the current gain setting as a percentage of the maximum achievable effect.

#### **LENGTH**

Returns the total length of the segment. The amplifier mixer device does not support this function.

#### **LENGTH TRACK number**

Returns the length of the track specified by **number**. The amplifier mixer device does not support this function.

#### **MONITOR**

Returns whether or not the amplifier mixer device is monitoring the input signal from one device while the output of another device is being recorded. Either TRUE or FALSE will be returned.

#### **NUMBER OF TRACKS**

Returns the number of tracks on the media. The amplifier mixer device does not support this function.

#### **PITCH**

Returns the current pitch setting as a percentage of the maximum achievable effect.

#### **POSITION**

Returns the current position. The amplifier mixer device does not support this function.

#### **POSITION IN TRACK**

Returns the current position relative to the beginning of the track. The amplifier mixer device does not support this function.

#### **POSITION TRACK number**

Returns the position of the start of the track specified by **number**. The amplifier mixer device does not support this function.

#### **SPEED FORMAT**

Returns the current speed format. The amplifier mixer device does not support this function.

#### **TIME FORMAT**

Returns the current time format. The amplifier mixer device does not support this function.

#### **TREBLE**

Returns the current treble setting as a percentage of the maximum achievable effect.

#### **VOLUME**

Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

#### **CONNECTOR type**

Specifies the connector that status information is desired from. The connector specified must be a valid connector.

#### **AUTO LEVEL CONTROL**

Returns the auto-level control setting for the desired connector.

#### **BALANCE**

Returns the balance setting for the desired connector.

#### **BASS**

Returns the bass setting for the desired connector.

#### **CHORUS**

Returns the chorus setting for the desired connector.

#### **CROSSOVER**

Returns the crossover setting for the desired connector.

#### **CUSTOM1**

Returns the custom effect setting for the desired connector.

#### **CUSTOM2**

Returns the custom effect setting for the desired connector.

#### **CUSTOM3**

Returns the custom effect setting for the desired connector.

#### **GAIN**

Returns the gain setting for the desired connector.

<b>LOUDNESS</b>	Returns the loudness setting for the desired connector.
<b>MID</b>	Returns the mid setting for the desired connector.
<b>MONITOR</b>	Returns the monitor setting for the desired connector.
<b>MUTE</b>	Returns the mute setting for the desired connector.
<b>PITCH</b>	Returns the pitch setting for the desired connector.
<b>REVERB</b>	Returns the reverb setting for the desired connector.
<b>TREBLE</b>	Returns the treble setting for the desired connector.
<b>VOLUME</b>	Returns the volume setting for the desired connector.
<b>WAIT</b>	The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified to receive return string information.
<b>NOTIFY</b>	The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an <a href="#">MM_MCINOTIFY</a> message is sent to the application window procedure.

## STATUS (Mixer Command) - Syntax Diagram

STATUS	object	AUDIO CHANNEL value BASS BALANCE CURRENT TRACK GAIN LENGTH LENGTH TRACK number MONITOR NUMBER OF TRACKS PITCH POSITION POSITION IN TRACK POSITION TRACK number SPEED FORMAT TIME FORMAT TREBLE VOLUME	
	CONNECTOR type	AUTO LEVEL CONTROL BALANCE BASS CHORUS CROSSOVER CUSTOM1 CUSTOM2 CUSTOM3 GAIN LOUDNESS MID MONITOR	WAIT NOTIFY

[MUTE](#)  
[PITCH](#)  
[REVERB](#)  
[TREBLE](#)  
[VOLUME](#)

Examples

---

## STATUS (Mixer Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## CD Audio Commands

The CD audio device supports the device-type specific command, [CUE](#), and extensions to the following basic and required commands:

- [CAPABILITY](#)
- [CONNECTOR](#)
- [CUE](#)
- [INFO](#)
- [PLAY](#)
- [SET](#)
- [STATUS](#)

---

## CAPABILITY

---

## CAPABILITY (CD Audio Command) - Example

The following command returns FALSE.

```
capability cdaudio can record wait
```

---

## CAPABILITY (CD Audio Command) - Purpose

The CAPABILITY command requests additional information about the capabilities of the CD device.

---

## CAPABILITY (CD Audio Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CAPABILITY (CD Audio Command) Keyword - CAN EJECT

### **CAN EJECT**

Returns TRUE if the CD audio device can eject the media.

---

## CAPABILITY (CD Audio Command) Keyword - CAN LOCKEJECT

### **CAN LOCKEJECT**

Returns TRUE if the device can disable manual ejection of the media.

---

## CAPABILITY (CD Audio Command) Keyword - CAN PLAY

### **CAN PLAY**

Returns TRUE if the CD audio device can play the media.

---

## CAPABILITY (CD Audio Command) Keyword - CAN PROCESS INTERNAL

### **CAN PROCESS INTERNAL**

Returns TRUE if the device can internally process digital data with an internal digital to analog converter (DAC).



---

## CAPABILITY (CD Audio Command) Keyword - CAN RECORD

### **CAN RECORD**

Returns FALSE. CD audio devices cannot record.

---

## CAPABILITY (CD Audio Command) Keyword - CAN SAVE

### **CAN SAVE**

Returns FALSE. CD audio devices cannot save data.

---

## CAPABILITY (CD Audio Command) Keyword - CAN SETVOLUME

### **CAN SETVOLUME**

Returns TRUE if the device supports software control of volume level.

---

## CAPABILITY (CD Audio Command) Keyword - CAN STREAM

### **CAN STREAM**

Returns TRUE if the device can continuously transfer digital data to or from another device. The source or destination of the data is determined by the device context connection.

---

## CAPABILITY (CD Audio Command) Keyword - COMPOUND DEVICE

### **COMPOUND DEVICE**

Returns FALSE. CD audio devices are simple devices.

---

# CAPABILITY (CD Audio Command) Keyword - DEVICE TYPE

**DEVICE TYPE**  
Returns **CDaudio**.

-----

# CAPABILITY (CD Audio Command) Keyword - HAS AUDIO

**HAS AUDIO**  
Returns **TRUE**.

-----

# CAPABILITY (CD Audio Command) Keyword - HAS VIDEO

**HAS VIDEO**  
Returns **FALSE**. CD audio devices do not support video.

-----

# CAPABILITY (CD Audio Command) Keyword - PREROLL TIME

**PREROLL TIME**  
Returns 0, indicating the preroll time is not bounded.

-----

# CAPABILITY (CD Audio Command) Keyword - PREROLL TYPE

**PREROLL TYPE**  
Returns the preroll characteristics of the device: Returns **NONE**.

-----

# CAPABILITY (CD Audio Command) Keyword - USES FILES

**USES FILES**

Returns FALSE. CD audio devices do not use files.

---

## CAPABILITY (CD Audio Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

---

## CAPABILITY (CD Audio Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CAPABILITY (CD Audio Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### **CAN EJECT**

Returns TRUE if the CD audio device can eject the media.

### **CAN LOCKEJECT**

Returns TRUE if the device can disable manual ejection of the media.

### **CAN PLAY**

Returns TRUE if the CD audio device can play the media.

### **CAN PROCESS INTERNAL**

Returns TRUE if the device can internally process digital data with an internal digital to analog converter (DAC).

### **CAN RECORD**

Returns FALSE. CD audio devices cannot record.

### **CAN SAVE**

Returns FALSE. CD audio devices cannot save data.

### **CAN SETVOLUME**

Returns TRUE if the device supports software control of volume level.

### **CAN STREAM**

Returns TRUE if the device can continuously transfer digital data to or from another device. The source or destination of the data is determined by the device context connection.

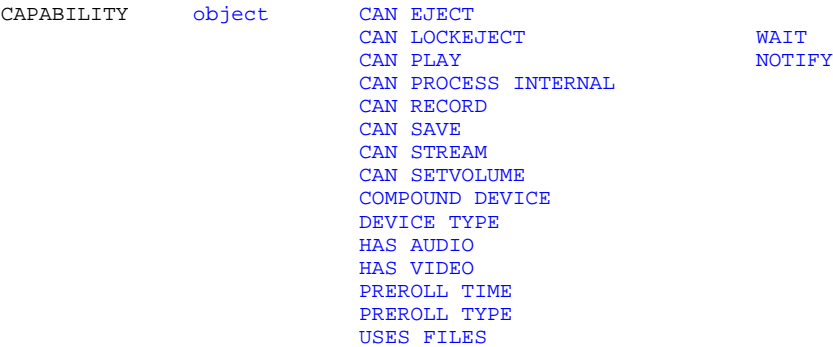
### **COMPOUND DEVICE**

Returns FALSE. CD audio devices are simple devices.

- DEVICE TYPE**  
Returns **CDaudio**.
- HAS AUDIO**  
Returns TRUE.
- HAS VIDEO**  
Returns FALSE. CD audio devices do not support video.
- PREROLL TIME**  
Returns 0, indicating the preroll time is not bounded.
- PREROLL TYPE**  
Returns the preroll characteristics of the device: Returns NONE.
- USES FILES**  
Returns FALSE. CD audio devices do not use files.
- WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.
- NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CAPABILITY (CD Audio Command) - Syntax Diagram



-----

## CAPABILITY (CD Audio Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

# CONNECTOR

---

## CONNECTOR (CD Audio Command) - Example

(This example assumes the device can stream.)

```
open cdaudio alias cdaud shareable wait
connector cdaud enable type CD stream wait
```

---

## CONNECTOR (CD Audio Command) - Purpose

The CONNECTOR command enables, disables, or queries the status of connectors on a device.

---

## CONNECTOR (CD Audio Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CONNECTOR (CD Audio Command) Keyword - ENABLE

### **ENABLE**

Enables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

---

## CONNECTOR (CD Audio Command) Keyword - DISABLE

### **DISABLE**

Disables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both

also be specified.

-----

## CONNECTOR (CD Audio Command) Keyword - QUERY

### QUERY

Queries the status of the indicated connector. The return value will be either "true" or "false" to indicate enabled or disabled. Using this option requires that the NUMBER or TYPE keywords, or both also be specified.

-----

## CONNECTOR (CD Audio Command) Keyword - NUMBER connector\_number

### NUMBER connector\_number

Indicates the connector number on which to perform the requested action. If this keyword is omitted, then the first connector is assumed. If the TYPE keyword is included, the connector number is interpreted as a relative offset within the specified connector type.

-----

## CONNECTOR (CD Audio Command) Keyword - TYPE connector\_type

### TYPE connector\_type

Indicates the type of connector to which the requested action applies. The following connector types are supported by this device:

CD stream

Digital output for CD devices. If this connector is enabled and the device has returned TRUE to the [CAPABILITY object CAN STREAM WAIT](#) command, the CD device will stream digital data to its associated amp/mixer.

headphones

The headphones connector. This connector is usually attached to a pair of headphones on the CD device itself.

-----

## CONNECTOR (CD Audio Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## CONNECTOR (CD Audio Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

## CONNECTOR (CD Audio Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**ENABLE**

Enables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

**DISABLE**

Disables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

## QUERY

Queries the status of the indicated connector. The return value will be either "true" or "false" to indicate enabled or disabled. Using this option requires that the NUMBER or TYPE keywords, or both also be specified.

**NUMBER** connector\_number

Indicates the connector number on which to perform the requested action. If this keyword is omitted, then the first connector is assumed. If the TYPE keyword is included, the connector number is interpreted as a relative offset within the specified connector type.

**TYPE** connector\_type

Indicates the type of connector to which the requested action applies. The following connector types are supported by this device:

CD stream

Digital output for CD devices. If this connector is enabled and the device has returned TRUE to the **CAPABILITY** object **CAN STREAM WAIT** command, the CD device will stream digital data to its associated amp/mixer.

headphones

The headphones connector. This connector is usually attached to a pair of headphones on the CD device itself.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

## CONNECTOR (CD Audio Command) - Syntax Diagram

CONNECTOR	object	ENABLE
		DISABLE
		QUERY

NUMBER connector\_number  
TYPE connector\_type

WAIT  
NOTIFY

## Examples

---

# CONNECTOR (CD Audio Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## CUE

---

# CUE (CD Audio Command) - Example

```
cue cdaudio output wait
```

---

# CUE (CD Audio Command) - Purpose

The CUE command prepares for playback. The CUE command does not have to be issued prior to playback; however, depending on the device, it might reduce the delay associated with the [PLAY](#) command.

The CUE command is not related to the [SETCUEPOINT](#) command.

---

# CUE (CD Audio Command) Keyword - object

## object

Object associated with this media control interface command. The object can be one of the following:



- Device type
- Device name
- Filename
- Alias

-----

## CUE (CD Audio Command) Keyword - OUTPUT

### OUTPUT

Prepares the device for playback.

-----

## CUE (CD Audio Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## CUE (CD Audio Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CUE (CD Audio Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### OUTPUT

Prepares the device for playback.

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CUE (CD Audio Command) - Syntax Diagram

CUE      *object*      OUTPUT      WAIT  
NOTIFY

Examples

---

## CUE (CD Audio Command) - Topics

Select an item:  
[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## INFO

---

## INFO (CD Audio Command) - Example

```
info cdaudio product wait
```

---

## INFO (CD Audio Command) - Purpose

The INFO command fills a user-supplied buffer with information.

---

## INFO (CD Audio Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## INFO (CD Audio Command) Keyword - ID

**ID**

Returns the disc ID if the device supports this function, otherwise, it returns 0. The value returned is a binary value.

-----

## INFO (CD Audio Command) Keyword - PRODUCT

**PRODUCT**

Returns the product name and model of the current audio device.

-----

## INFO (CD Audio Command) Keyword - UPC

**UPC**

Returns the disc UPC code (serial number) if the device supports this function, otherwise, it returns 0. The value returned is a binary value.

-----

## INFO (CD Audio Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## INFO (CD Audio Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## INFO (CD Audio Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### ID

Returns the disc ID if the device supports this function, otherwise, it returns 0. The value returned is a binary value.

### PRODUCT

Returns the product name and model of the current audio device.

### UPC

Returns the disc UPC code (serial number) if the device supports this function, otherwise, it returns 0. The value returned is a binary value.

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## INFO (CD Audio Command) - Syntax Diagram

INFO      [object](#)      [ID](#)  
                                 [PRODUCT](#)  
                                 [UPC](#)      [WAIT](#)  
   [NOTIFY](#)



-----

## INFO (CD Audio Command) - Topics

Select an item:

- [Purpose](#)
- [Syntax Diagram](#)
- [Keywords](#)
- [Example](#)
- [Glossary](#)

---

## PLAY

---

## PLAY (CD Audio Command) - Example

```
play cdaudio from 10000 to 30000 wait
```

---

## PLAY (CD Audio Command) - Purpose

The PLAY command starts playing audio.

---

## PLAY (CD Audio Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## PLAY (CD Audio Command) Keyword - FROM pos

### **FROM pos**

The position to start playing. If FROM is omitted, the device starts playing at the current position. If the FROM position is greater than the end position of the disc, or if the FROM position is greater than the TO position, an error is returned.

---

## PLAY (CD Audio Command) Keyword - TO pos

### **TO pos**

The position to stop playing. If TO is omitted, play stops at the end of the disc. If the TO position is greater than the length of the disc, it receives an OUT OF RANGE error.

---

## PLAY (CD Audio Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## PLAY (CD Audio Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

---

## PLAY (CD Audio Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### **FROM pos**

The position to start playing. If FROM is omitted, the device starts playing at the current position. If the FROM position is greater than the end position of the disc, or if the FROM position is greater than the TO position, an error is returned.

### **TO pos**

The position to stop playing. If TO is omitted, play stops at the end of the disc. If the TO position is greater than the length of the disc, it receives an OUT OF RANGE error.

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

---

## PLAY (CD Audio Command) - Syntax Diagram

PLAY      *object*

FROM pos      WAIT  
TO pos      NOTIFY

## Examples

# PLAY (CD Audio Command) - Topics

Select an item:

- [Purpose](#)
- [Syntax Diagram](#)
- [Keywords](#)
- [Example](#)
- [Glossary](#)

## SET

# SET (CD Audio Command) - Example

```
set cdaudio time format tmsf wait
```

# SET (CD Audio Command) - Purpose

The SET command sets the various control items.

# SET (CD Audio Command) Keyword - object

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type

- Device name
- Filename
- Alias

-----

## SET (CD Audio Command) Keyword - AUDIO

### AUDIO

Sets the audio attributes of the device context specified by the ALL, LEFT, RIGHT, OVER, and VOLUME keywords.

-----

## SET (CD Audio Command) Keyword - ALL

### ALL

Applies to both/all channels (default).

Specify ON or OFF with the ALL keyword.

-----

## SET (CD Audio Command) Keyword - ON

### ON

Enables audio output.

-----

## SET (CD Audio Command) Keyword - OFF

### OFF

Disables audio output.

-----

## SET (CD Audio Command) Keyword - LEFT

### LEFT

Applies to left channel.

Specify ON or OFF with the LEFT keyword.

-----



# SET (CD Audio Command) Keyword - ON

**ON**  
Enables audio output to the left channel.

-----

# SET (CD Audio Command) Keyword - OFF

**OFF**  
Disables audio output to the left channel.

-----

# SET (CD Audio Command) Keyword - RIGHT

**RIGHT**  
Applies to right channel.  
Specify ON or OFF with the RIGHT keyword.

-----

# SET (CD Audio Command) Keyword - ON

**ON**  
Enables audio output to the right channel.

-----

# SET (CD Audio Command) Keyword - OFF

**OFF**  
Disables audio output to the right channel.

-----

# SET (CD Audio Command) Keyword - OVER milliseconds

**OVER milliseconds**  
Applies the change over the specified time period (fade).

---

## SET (CD Audio Command) Keyword - VOLUME percentage

### **VOLUME percentage**

Sets the device/mixer channel volume level.

---

## SET (CD Audio Command) Keyword - DOOR CLOSED

### **DOOR CLOSED**

Retracts the tray and closes the door, if possible.

---

## SET (CD Audio Command) Keyword - DOOR LOCKED

### **DOOR LOCKED**

Locks the media cover on the device (if any). This disables manual ejection of the media from the device.

---

## SET (CD Audio Command) Keyword - DOOR OPEN

### **DOOR OPEN**

Opens the door and ejects the tray, if possible. This enables manual ejection of the media from the device.

---

## SET (CD Audio Command) Keyword - DOOR UNLOCKED

### **DOOR UNLOCKED**

Unlocks the media cover on the device (if any). This enables manual ejection of the media from the device.

---

## SET (CD Audio Command) Keyword - TIME FORMAT MILLISECONDS

### **TIME FORMAT MILLISECONDS**

Sets the time format to milliseconds. All position information is in this format after this command. You can abbreviate as **ms**.

-----

## SET (CD Audio Command) Keyword - TIME FORMAT MMTIME

### TIME FORMAT MMTIME

Sets the time format to MMTIME.

-----

## SET (CD Audio Command) Keyword - TIME FORMAT MSF

### TIME FORMAT MSF

Sets the time format to *mm:ss:ff*, where *mm* is minutes, *ss* is seconds, and *ff* is frames. All position information is in this format after this command. The *ff* input can be omitted if it is 0; *ss* can be omitted if both *ss* and *ff* are 0. These fields have the following maximum values:

Minutes ( <i>mm</i> )	99
Seconds ( <i>ss</i> )	59
Frames ( <i>ff</i> )	74

-----

## SET (CD Audio Command) Keyword - TIME FORMAT TMSF

### TIME FORMAT TMSF

Sets the time format to *tt:mm:ss:ff* where *tt* is tracks, *mm* is minutes, *ss* is seconds, and *ff* is frames. All position information is in this format after this command. The *ff* input can be omitted if it is 0, *ss* can be omitted if both *ss* and *ff* are 0, and *mm* can be omitted if *mm*, *ss*, and *ff* all equal 0. These fields have the following maximum values:

Tracks ( <i>tt</i> )	99
Minutes ( <i>mm</i> )	99
Seconds ( <i>ss</i> )	59
Frames ( <i>ff</i> )	74

-----

## SET (CD Audio Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

# SET (CD Audio Command) Keyword - NOTIFY

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# SET (CD Audio Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## AUDIO

Sets the audio attributes of the device context specified by the ALL, LEFT, RIGHT, OVER, and VOLUME keywords.

### ALL

Applies to both/all channels (default).

Specify ON or OFF with the ALL keyword.

#### ON

Enables audio output.

#### OFF

Disables audio output.

### LEFT

Applies to left channel.

Specify ON or OFF with the LEFT keyword.

#### ON

Enables audio output to the left channel.

#### OFF

Disables audio output to the left channel.

### RIGHT

Applies to right channel.

Specify ON or OFF with the RIGHT keyword.

#### ON

Enables audio output to the right channel.

#### OFF

Disables audio output to the right channel.

### OVER milliseconds

Applies the change over the specified time period (fade).

### VOLUME percentage

Sets the device/mixer channel volume level.

## DOOR CLOSED

Retracts the tray and closes the door, if possible.

**DOOR LOCKED**

Locks the media cover on the device (if any). This disables manual ejection of the media from the device.

**DOOR OPEN**

Opens the door and ejects the tray, if possible. This enables manual ejection of the media from the device.

**DOOR UNLOCKED**

Unlocks the media cover on the device (if any). This enables manual ejection of the media from the device.

**TIME FORMAT MILLISECONDS**

Sets the time format to milliseconds. All position information is in this format after this command. You can abbreviate as **ms**.

**TIME FORMAT MMTIME**

Sets the time format to MMTIME.

**TIME FORMAT MSF**

Sets the time format to *mm:ss:ff*, where *mm* is minutes, *ss* is seconds, and *ff* is frames. All position information is in this format after this command. The *ff* input can be omitted if it is 0; *ss* can be omitted if both *ss* and *ff* are 0. These fields have the following maximum values:

Minutes ( <i>mm</i> )	99
Seconds ( <i>ss</i> )	59
Frames ( <i>ff</i> )	74

**TIME FORMAT TMSF**

Sets the time format to *tt:mm:ss:ff* where *tt* is tracks, *mm* is minutes, *ss* is seconds, and *ff* is frames. All position information is in this format after this command. The *ff* input can be omitted if it is 0, *ss* can be omitted if both *ss* and *ff* are 0, and *mm* can be omitted if *mm*, *ss*, and *ff* all equal 0. These fields have the following maximum values:

Tracks ( <i>tt</i> )	99
Minutes ( <i>mm</i> )	99
Seconds ( <i>ss</i> )	59
Frames ( <i>ff</i> )	74

**WAIT**

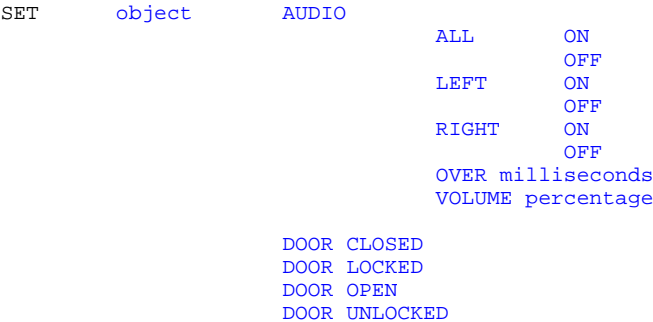
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

-----

# SET (CD Audio Command) - Syntax Diagram



[TIME FORMAT MILLISECONDS](#)  
[TIME FORMAT MMTIME](#)  
[TIME FORMAT MSF](#)  
[TIME FORMAT TMSF](#)

[WAIT](#)  
[NOTIFY](#)

## Examples

---

# SET (CD Audio Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

# STATUS

---

# STATUS (CD Audio Command) - Example

```
status cdaudio mode wait
```

---

# STATUS (CD Audio Command) - Purpose

The STATUS command obtains status information for the device.

---

# STATUS (CD Audio Command) Keyword - object

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type

- Device name
- Filename
- Alias

-----

## STATUS (CD Audio Command) Keyword - CHANNELS

### **CHANNELS**

Returns the number of audio channels on the current track.

-----

## STATUS (CD Audio Command) Keyword - CHANNELS TRACK number

### **CHANNELS TRACK number**

Returns the number of audio channels on the track.

-----

## STATUS (CD Audio Command) Keyword - COPYPERMITTED

### **COPYPERMITTED**

Returns TRUE if digital copying is permitted.

-----

## STATUS (CD Audio Command) Keyword - COPYPERMITTED TRACK number

### **COPYPERMITTED TRACK number**

Returns TRUE if digital copying is permitted.

-----

## STATUS (CD Audio Command) Keyword - CURRENT TRACK

### **CURRENT TRACK**

Returns the current track.

-----

# STATUS (CD Audio Command) Keyword - LENGTH

**LENGTH**  
Returns the total length of the disc.

-----

# STATUS (CD Audio Command) Keyword - LENGTH TRACK number

**LENGTH TRACK number**  
Returns the length of the track specified by **number**.

-----

# STATUS (CD Audio Command) Keyword - MEDIA PRESENT

**MEDIA PRESENT**  
Returns TRUE if the media is inserted in the device; otherwise, it returns FALSE.

-----

# STATUS (CD Audio Command) Keyword - MODE

**MODE**  
Returns the current mode of the device: **not ready**, **stopped**, **playing**, **seeking**, **recording**, **paused**, or **other**.

-----

# STATUS (CD Audio Command) Keyword - NUMBER OF TRACKS

**NUMBER OF TRACKS**  
Returns the number of tracks.

-----

# STATUS (CD Audio Command) Keyword - POSITION



#### **POSITION**

Returns the current position.

-----

## STATUS (CD Audio Command) Keyword - POSITION IN TRACK

#### **POSITION IN TRACK**

Returns the current position relative to the beginning of the current track.

-----

## STATUS (CD Audio Command) Keyword - POSITION TRACK number

#### **POSITION TRACK number**

Returns the starting position of the track specified by **number**.

-----

## STATUS (CD Audio Command) Keyword - PREEMPHASIS

#### **PREEMPHASIS**

Returns TRUE if the current track was recorded with pre-emphasis.

-----

## STATUS (CD Audio Command) Keyword - PREEMPHASIS TRACK number

#### **PREEMPHASIS TRACK number**

Returns TRUE if the specified track was recorded with pre-emphasis.

-----

## STATUS (CD Audio Command) Keyword - READY

#### **READY**

Returns TRUE if the device is ready.

---

## STATUS (CD Audio Command) Keyword - START POSITION

### START POSITION

Returns the starting position of the media.

---

## STATUS (CD Audio Command) Keyword - TIME FORMAT

### TIME FORMAT

Returns the time format.

---

## STATUS (CD Audio Command) Keyword - TYPE

### TYPE

Returns the track type of the current track. The track type will be returned as one of the following values:

- audio
  - data
  - other
- 

## STATUS (CD Audio Command) Keyword - TYPE TRACK number

### TYPE TRACK number

Returns the track type of the specified track. If the track number is not specified the current track is assumed. The track type will be returned as one of the following values:

- audio
  - data
  - other
- 

## STATUS (CD Audio Command) Keyword - VOLUME

### VOLUME

Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of

the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

-----

## STATUS (CD Audio Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

-----

## STATUS (CD Audio Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

-----

## STATUS (CD Audio Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**Note:** If **TRACK number** is used with the keywords TYPE, COPYPERMITTED, CHANNELS, or PREEMPHASIS, then the specified track is referenced, otherwise, the current track is referenced.

### **CHANNELS**

Returns the number of audio channels on the current track.

### **CHANNELS TRACK number**

Returns the number of audio channels on the track.

### **COPYPERMITTED**

Returns TRUE if digital copying is permitted.

### **COPYPERMITTED TRACK number**

Returns TRUE if digital copying is permitted.

### **CURRENT TRACK**

Returns the current track.

### **LENGTH**

Returns the total length of the disc.

### **LENGTH TRACK number**

Returns the length of the track specified by **number**.

**MEDIA PRESENT**

Returns TRUE if the media is inserted in the device; otherwise, it returns FALSE.

**MODE**

Returns the current mode of the device: **not ready**, **stopped**, **playing**, **seeking**, **recording**, **paused**, or **other**.

**NUMBER OF TRACKS**

Returns the number of tracks.

**POSITION**

Returns the current position.

**POSITION IN TRACK**

Returns the current position relative to the beginning of the current track.

**POSITION TRACK number**

Returns the starting position of the track specified by **number**.

**PREEMPHASIS**

Returns TRUE if the current track was recorded with pre-emphasis.

**PREEMPHASIS TRACK number**

Returns TRUE if the specified track was recorded with pre-emphasis.

**READY**

Returns TRUE if the device is ready.

**START POSITION**

Returns the starting position of the media.

**TIME FORMAT**

Returns the time format.

**TYPE**

Returns the track type of the current track. The track type will be returned as one of the following values:

- audio
- data
- other

**TYPE TRACK number**

Returns the track type of the specified track. If the track number is not specified the current track is assumed. The track type will be returned as one of the following values:

- audio
- data
- other

**VOLUME**

Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## STATUS (CD Audio Command) - Syntax Diagram

STATUS

[object](#)

[CHANNELS](#)

CHANNELS TRACK number	WAIT
COPYPERMITTED	NOTIFY
COPYPERMITTED TRACK number	
CURRENT TRACK	
LENGTH	
LENGTH TRACK number	
MEDIA PRESENT	
MODE	
NUMBER OF TRACKS	
POSITION	
POSITION IN TRACK	
POSITION TRACK number	
PREEMPHASIS	
PREEMPHASIS TRACK number	
READY	
START POSITION	
TIME FORMAT	
TYPE	
TYPE TRACK number	
VOLUME	

## Examples

---

# STATUS (CD Audio Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## CD/XA Commands

The CD/XA audio device supports the device-type specific command, [CUE](#), and extensions to the following basic and required commands:

- [CAPABILITY](#)
- [CONNECTOR](#)
- [CUE](#)
- [INFO](#)
- [PLAY](#)
- [SEEK](#)
- [SET](#)
- [STATUS](#)

---

## CAPABILITY

---

# CAPABILITY (CD/XA Command) - Example

capability cdxa can record wait

---

## CAPABILITY (CD/XA Command) - Purpose

The CAPABILITY command requests additional information about the capabilities of the CD/XA device.

---

## CAPABILITY (CD/XA Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CAPABILITY (CD/XA Command) Keyword - CAN EJECT

### **CAN EJECT**

Returns TRUE if the CD/XA device can eject the media.

---

## CAPABILITY (CD/XA Command) Keyword - CAN LOCKEJECT

### **CAN LOCKEJECT**

Returns TRUE if the device can disable manual ejection of the media.

---

## CAPABILITY (CD/XA Command) Keyword - CAN PLAY

### **CAN PLAY**

Returns TRUE if the CD/XA device can play the media.

---

## CAPABILITY (CD/XA Command) Keyword - CAN PROCESS INTERNAL

### **CAN PROCESS INTERNAL**

Returns TRUE if the device can internally process digital data with an internal digital to analog converter (DAC).

---

## CAPABILITY (CD/XA Command) Keyword - CAN RECORD

### **CAN RECORD**

Returns FALSE. CD/XA devices cannot record.

---

## CAPABILITY (CD/XA Command) Keyword - CAN SAVE

### **CAN SAVE**

Returns FALSE. CD/XA devices cannot save data.

---

## CAPABILITY (CD/XA Command) Keyword - CAN SETVOLUME

### **CAN SETVOLUME**

Returns TRUE if the device supports software control of volume level.

---

## CAPABILITY (CD/XA Command) Keyword - CAN STREAM

### **CAN STREAM**

Returns TRUE if the device can continuously transfer digital data to or from another device. The source or destination of the data is determined by the device context connection.

---

## CAPABILITY (CD/XA Command) Keyword - COMPOUND

# DEVICE

**COMPOUND DEVICE**  
Returns TRUE. CD/XA devices are compound devices and utilize files.

-----

## CAPABILITY (CD/XA Command) Keyword - DEVICE TYPE

**DEVICE TYPE**  
Returns CDXA.

-----

## CAPABILITY (CD/XA Command) Keyword - HAS AUDIO

**HAS AUDIO**  
Returns TRUE.

-----

## CAPABILITY (CD/XA Command) Keyword - HAS VIDEO

**HAS VIDEO**  
Returns FALSE if no video available. CD/XA devices do not support video for this release of the Toolkit.

-----

## CAPABILITY (CD/XA Command) Keyword - PREROLL TIME

**PREROLL TIME**  
Returns 0, indicating the preroll time is not bounded.

-----

## CAPABILITY (CD/XA Command) Keyword - PREROLL TYPE

**PREROLL TYPE**  
Returns the preroll characteristics of the device: Returns NONE.

-----



# CAPABILITY (CD/XA Command) Keyword - USES FILES

## USES FILES

Returns TRUE.

---

# CAPABILITY (CD/XA Command) Keyword - WAIT

## WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

---

# CAPABILITY (CD/XA Command) Keyword - NOTIFY

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# CAPABILITY (CD/XA Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## CAN EJECT

Returns TRUE if the CD/XA device can eject the media.

## CAN LOCKEJECT

Returns TRUE if the device can disable manual ejection of the media.

## CAN PLAY

Returns TRUE if the CD/XA device can play the media.

## CAN PROCESS INTERNAL

Returns TRUE if the device can internally process digital data with an internal digital to analog converter (DAC).

## CAN RECORD

Returns FALSE. CD/XA devices cannot record.

## CAN SAVE

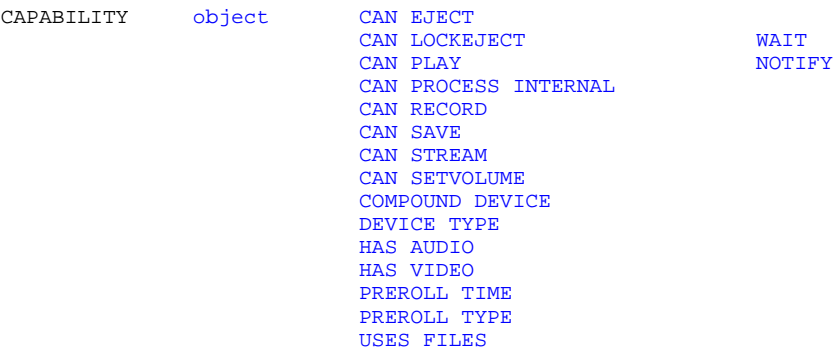
Returns FALSE. CD/XA devices cannot save data.

## CAN SETVOLUME

Returns TRUE if the device supports software control of volume level.

- CAN STREAM**  
Returns TRUE if the device can continuously transfer digital data to or from another device. The source or destination of the data is determined by the device context connection.
- COMPOUND DEVICE**  
Returns TRUE. CD/XA devices are compound devices and utilize files.
- DEVICE TYPE**  
Returns CDXA.
- HAS AUDIO**  
Returns TRUE.
- HAS VIDEO**  
Returns FALSE if no video available. CD/XA devices do not support video for this release of the Toolkit.
- PREROLL TIME**  
Returns 0, indicating the preroll time is not bounded.
- PREROLL TYPE**  
Returns the preroll characteristics of the device: Returns NONE.
- USES FILES**  
Returns TRUE.
- WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.
- NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

## CAPABILITY (CD/XA Command) - Syntax Diagram



## CAPABILITY (CD/XA Command) - Topics

Select an item:

---

## CONNECTOR

---

### CONNECTOR (CD/XA Command) - Example

```
connector cdx query type XA stream wait
```

---

### CONNECTOR (CD/XA Command) - Purpose

The CONNECTOR command enables, disables, or queries the status of connectors on a device.

---

### CONNECTOR (CD/XA Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

### CONNECTOR (CD/XA Command) Keyword - ENABLE

**ENABLE**

Enables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

---

### CONNECTOR (CD/XA Command) Keyword - DISABLE

**DISABLE**  
Disables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

-----

## CONNECTOR (CD/XA Command) Keyword - QUERY

**QUERY**  
Queries the status of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

-----

## CONNECTOR (CD/XA Command) Keyword - NUMBER connector\_number

**NUMBER connector\_number**  
Indicates the connector number on which to perform the requested action. If this item is omitted, then the first connector is assumed. If the TYPE keyword is included, the connector number is interpreted as a relative offset within the specified connector type.

-----

## CONNECTOR (CD/XA Command) Keyword - TYPE connector\_type

**TYPE connector\_type**  
Indicates the type of connector to which the requested action applies. The following connector types are supported by this device.

XA stream	Digital output for the audio amplifier/mixer. This connector is always enabled.
-----------	---

-----

## CONNECTOR (CD/XA Command) Keyword - WAIT

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

-----

## CONNECTOR (CD/XA Command) Keyword - NOTIFY

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

-----

## CONNECTOR (CD/XA Command) - Keywords

#### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

#### ENABLE

Enables information flow through the indicated connector. Use of this option requires that the **NUMBER** or **TYPE** keywords, or both also be specified.

#### DISABLE

Disables information flow through the indicated connector. Use of this option requires that the **NUMBER** or **TYPE** keywords, or both also be specified.

#### QUERY

Queries the status of the indicated connector. The return value will be either **TRUE** or **FALSE** to indicate enabled or disabled. Use of this option requires that the **NUMBER** or **TYPE** keywords, or both also be specified.

#### NUMBER connector\_number

Indicates the connector number on which to perform the requested action. If this item is omitted, then the first connector is assumed. If the **TYPE** keyword is included, the connector number is interpreted as a relative offset within the specified connector type.

#### TYPE connector\_type

Indicates the type of connector to which the requested action applies. The following connector types are supported by this device.

XA stream

Digital output for the audio amplifier/mixer. This connector is always enabled.

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The **WAIT** keyword must be specified in order to receive return string information.

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

-----

## CONNECTOR (CD/XA Command) - Syntax Diagram

```
CONNECTOR      object      ENABLE
                  DISABLE
                  QUERY

NUMBER connector_number
TYPE connector_type

WAIT
NOTIFY
```

## Examples

---

# CONNECTOR (CD/XA Command) - Topics

Select an item:

[Purpose](#)

[Syntax Diagram](#)

[Keywords](#)

[Example](#)

[Glossary](#)

---

## CUE

---

# CUE (CD/XA Command) - Example

```
open cdx a alias cd shareable wait
load cd \brazil\carnival.xa wait
cue cd output wait
```

---

# CUE (CD/XA Command) - Purpose

The CUE command prepares for playback. The CUE command does not have to be issued prior to playback; however, depending on the device, it might reduce the delay associated with the CUE command.

The CUE command is not related to the [SETCUEPOINT](#) command.

---

# CUE (CD/XA Command) Keyword - object

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename

- Alias

-----

## CUE (CD/XA Command) Keyword - OUTPUT

### OUTPUT

Cues the device for playback.

-----

## CUE (CD/XA Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## CUE (CD/XA Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CUE (CD/XA Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### OUTPUT

Cues the device for playback.

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CUE (CD/XA Command) - Syntax Diagram

CUE      [object](#)      OUTPUT      [WAIT](#)  
NOTIFY

**Examples**

CUE (CD/XA Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

INFO

INFO (CD/XA Command) - Example

```
info cdx a product wait
```

INFO (CD/XA Command) - Purpose

The INFO command fills a user-supplied buffer with information.

INFO (CD/XA Command) Keyword - object

**object**  
Object associated with this media control interface command. The object can be one of the following:



- Device type
- Device name
- Filename
- Alias

-----

## INFO (CD/XA Command) Keyword - ID

### ID

Returns the disc ID if the device supports this function, otherwise, it returns 0. The value returned is a binary value.

-----

## INFO (CD/XA Command) Keyword - PRODUCT

### PRODUCT

Returns the product name and model of the current audio device.

-----

## INFO (CD/XA Command) Keyword - UPC

### UPC

Returns the disc UPC code (serial number) if the device supports this function, otherwise, it returns 0. The value returned is a binary value.

-----

## INFO (CD/XA Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## INFO (CD/XA Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# INFO (CD/XA Command) - Keywords

- object**  
Object associated with this media control interface command. The object can be one of the following:
- Device type
  - Device name
  - Filename
  - Alias
- ID**  
Returns the disc ID if the device supports this function, otherwise, it returns 0. The value returned is a binary value.
- PRODUCT**  
Returns the product name and model of the current audio device.
- UPC**  
Returns the disc UPC code (serial number) if the device supports this function, otherwise, it returns 0. The value returned is a binary value.
- WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.
- NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## INFO (CD/XA Command) - Syntax Diagram

INFO      [object](#)      [ID](#)  
                                 [PRODUCT](#)  
                                 [UPC](#)      [WAIT](#)  
   [NOTIFY](#)



---

## INFO (CD/XA Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

PLAY

---

## PLAY (CD/XA Command) - Example

```
play cdx a from 10000 to 50000 wait
```

---

## PLAY (CD/XA Command) - Purpose

The PLAY command starts playing audio.

---

## PLAY (CD/XA Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## PLAY (CD/XA Command) Keyword - FROM pos

### **FROM pos**

The position to start playing. If FROM is omitted, the device starts playing at the current position. If the FROM position is greater than the end position of the disc, or if the FROM position is greater than the TO position, an error is returned.

---

## PLAY (CD/XA Command) Keyword - TO pos

### **TO pos**

The position to stop playing. If TO is omitted, play stops at the end of the disc. If the TO position is greater than the length of the disc, it receives an MCIERR\_OUTOFRANGE error.

---

## PLAY (CD/XA Command) Keyword - WAIT

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## PLAY (CD/XA Command) Keyword - NOTIFY

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## PLAY (CD/XA Command) - Keywords

#### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

#### FROM pos

The position to start playing. If FROM is omitted, the device starts playing at the current position. If the FROM position is greater than the end position of the disc, or if the FROM position is greater than the TO position, an error is returned.

#### TO pos

The position to stop playing. If TO is omitted, play stops at the end of the disc. If the TO position is greater than the length of the disc, it receives an MCIERR\_OUTOFRANGE error.

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## PLAY (CD/XA Command) - Syntax Diagram

PLAY      [object](#)

            FROM pos      WAIT

            TO pos         NOTIFY

## Examples

---

# PLAY (CD/XA Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## SEEK

---

# SEEK (CD/XA Command) - Example

```
open cdx a alias cd shareable wait
load cd music.xa wait
seek cd to end wait
```

---

# SEEK (CD/XA Command) - Purpose

The SEEK command finds the specified location on the disc.

**Note:** Seeking to a given position in a CD-XA file requires the media control driver to essentially play to that point from the beginning of the file.

---

# SEEK (CD/XA Command) Keyword - object

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## SEEK (CD/XA Command) Keyword - TO pos

### **TO pos**

The destination position for the seek. If it is greater than the length of the disc, an MCIERR\_OUTOFRANGE error is returned.

-----

## SEEK (CD/XA Command) Keyword - TO START

### **TO START**

Seeks to the first playable location on the disc.

-----

## SEEK (CD/XA Command) Keyword - TO END

### **TO END**

Seeks to the end of the disc.

-----

## SEEK (CD/XA Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## SEEK (CD/XA Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## SEEK (CD/XA Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**TO pos**

The destination position for the seek. If it is greater than the length of the disc, an MCIERR\_OUTOFRANGE error is returned.

**TO START**

Seeks to the first playable location on the disc.

**TO END**

Seeks to the end of the disc.

**WAIT**

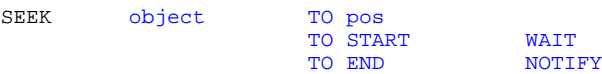
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# SEEK (CD/XA Command) - Syntax Diagram



---

# SEEK (CD/XA Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

# SET

---

## SET (CD/XA Command) - Example

```
set cdxa door open wait
```

---

## SET (CD/XA Command) - Purpose

The SET command sets the various control items.

---

## SET (CD/XA Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## SET (CD/XA Command) Keyword - AUDIO

### **AUDIO**

Sets the audio attributes of the device context specified by the ALL, LEFT, RIGHT, OVER, and VOLUME keywords.

---

## SET (CD/XA Command) Keyword - ALL

### **ALL**

Applies to both/all channels (default).

Specify ON or OFF with the ALL keyword.

---

## SET (CD/XA Command) Keyword - ON

### **ON**



Enables audio output.

---

## SET (CD/XA Command) Keyword - OFF

### OFF

Disables audio output.

---

## SET (CD/XA Command) Keyword - LEFT

### LEFT

Applies to left channel.

Specify ON or OFF with the LEFT keyword.

---

## SET (CD/XA Command) Keyword - ON

### ON

Enables audio output to the left channel.

---

## SET (CD/XA Command) Keyword - OFF

### OFF

Disables audio output to the left channel.

---

## SET (CD/XA Command) Keyword - RIGHT

### RIGHT

Applies to right channel.

Specify ON or OFF with the RIGHT keyword.

---

## SET (CD/XA Command) Keyword - ON

**ON**

Enables audio output to the right channel.

-----

## SET (CD/XA Command) Keyword - OFF

**OFF**

Disables audio output to the right channel.

-----

## SET (CD/XA Command) Keyword - OVER milliseconds

**OVER milliseconds**

Applies the change over the specified time period (fade).

-----

## SET (CD/XA Command) Keyword - VOLUME percentage

**VOLUME percentage**

Sets the device/mixer channel volume level.

-----

## SET (CD/XA Command) Keyword - CHANNEL number

**CHANNEL number**

Channel number to set for the given device.

-----

## SET (CD/XA Command) Keyword - AUDIO DEVICE

**AUDIO DEVICE**

Channel set pertains to the audio device.

-----

## SET (CD/XA Command) Keyword - DOOR CLOSED

**DOOR CLOSED**

Retracts the tray and closes the door, if possible.

---

## SET (CD/XA Command) Keyword - DOOR LOCKED

**DOOR LOCKED**

Locks the media cover on the device (if any). This disables manual ejection of the media from the device.

---

## SET (CD/XA Command) Keyword - DOOR OPEN

**DOOR OPEN**

Opens the door and ejects the tray, if possible. This enables manual ejection of the media from the device.

---

## SET (CD/XA Command) Keyword - DOOR UNLOCKED

**DOOR UNLOCKED**

Unlocks the media cover on the device (if any). This enables manual ejection of the media from the device.

---

## SET (CD/XA Command) Keyword - TIME FORMAT MILLISECONDS

**TIME FORMAT MILLISECONDS**

Sets time format, to milliseconds. All position information is in this format after this command. You can abbreviate milliseconds as **ms**.

---

## SET (CD/XA Command) Keyword - TIME FORMAT MMTIME

**TIME FORMAT MMTIME**

The time format is set to MMTIME.

---

# SET (CD/XA Command) Keyword - TIME FORMAT MSF

## TIME FORMAT MSF

Sets the time format to *mm:ss:ff*, where *mm* is minutes, *ss* is seconds, and *ff* is frames. All position information is in this format after this command.

-----

# SET (CD/XA Command) Keyword - TIME FORMAT TMSF

## TIME FORMAT TMSF

Sets the time format to *tt:mm:ss:ff* where *tt* is tracks, *mm* is minutes, *ss* is seconds, and *ff* is frames. All position information is in this format after this command.

-----

# SET (CD/XA Command) Keyword - WAIT

## WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

# SET (CD/XA Command) Keyword - NOTIFY

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# SET (CD/XA Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## AUDIO

Sets the audio attributes of the device context specified by the ALL, LEFT, RIGHT, OVER, and VOLUME keywords.

### ALL

Applies to both/all channels (default).

Specify ON or OFF with the ALL keyword.

**ON**  
Enables audio output.

**OFF**  
Disables audio output.

#### **LEFT**

Applies to left channel.

Specify ON or OFF with the LEFT keyword.

**ON**  
Enables audio output to the left channel.

**OFF**  
Disables audio output to the left channel.

#### **RIGHT**

Applies to right channel.

Specify ON or OFF with the RIGHT keyword.

**ON**  
Enables audio output to the right channel.

**OFF**  
Disables audio output to the right channel.

**OVER milliseconds**  
Applies the change over the specified time period (fade).

**VOLUME percentage**  
Sets the device/mixer channel volume level.

**CHANNEL number**  
Channel number to set for the given device.

**AUDIO DEVICE**  
Channel set pertains to the audio device.

**DOOR CLOSED**  
Retracts the tray and closes the door, if possible.

**DOOR LOCKED**  
Locks the media cover on the device (if any). This disables manual ejection of the media from the device.

**DOOR OPEN**  
Opens the door and ejects the tray, if possible. This enables manual ejection of the media from the device.

**DOOR UNLOCKED**  
Unlocks the media cover on the device (if any). This enables manual ejection of the media from the device.

**TIME FORMAT MILLISECONDS**  
Sets time format, to milliseconds. All position information is in this format after this command. You can abbreviate milliseconds as **ms**.

**TIME FORMAT MMTIME**  
The time format is set to MMTIME.

**TIME FORMAT MSF**  
Sets the time format to *mm:ss:ff*, where *mm* is minutes, *ss* is seconds, and *ff* is frames. All position information is in this format after this command.

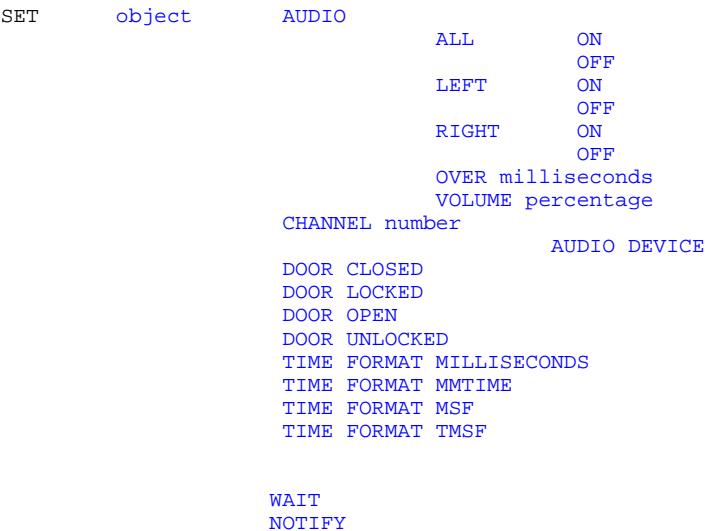
**TIME FORMAT TMSF**  
Sets the time format to *tt:mm:ss:ff* where *tt* is tracks, *mm* is minutes, *ss* is seconds, and *ff* is frames. All position information is in this format after this command.

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

SET (CD/XA Command) - Syntax Diagram



Examples

SET (CD/XA Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

STATUS

STATUS (CD/XA Command) - Example

status cdx mode wait

---

## STATUS (CD/XA Command) - Purpose

The STATUS command obtains status information for the device.

---

## STATUS (CD/XA Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## STATUS (CD/XA Command) Keyword - CHANNEL number

### CHANNEL number

Returns **audio device**, **audio buffer**, **video buffer**, **data buffer**, **none**, **right**, **all**, or **left** for the destination of the data in the channel specified by **number**.

---

## STATUS (CD/XA Command) Keyword - LENGTH

### LENGTH

Returns MCIERR\_INDETERMINATE\_LENGTH. The length of a CD-XA file cannot be determined in this release of OS/2 multimedia.

---

## STATUS (CD/XA Command) Keyword - MEDIA PRESENT

### MEDIA PRESENT

Returns TRUE if the media is inserted in the device; otherwise, it returns FALSE.

---

## STATUS (CD/XA Command) Keyword - MODE

### MODE

Returns the current mode of the device: **not ready**, **stopped**, **playing**, **seeking**, **recording**, **paused**, or **other**.

-----

## STATUS (CD/XA Command) Keyword - POSITION

### POSITION

Returns the current position.

-----

## STATUS (CD/XA Command) Keyword - READY

### READY

Returns TRUE if the device is ready.

-----

## STATUS (CD/XA Command) Keyword - TIME FORMAT

### TIME FORMAT

Returns the time format.

-----

## STATUS (CD/XA Command) Keyword - VOLUME

### VOLUME

Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

-----

## STATUS (CD/XA Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the



application. The WAIT keyword must be specified in order to receive return string information.

---

## STATUS (CD/XA Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

---

## STATUS (CD/XA Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### CHANNEL number

Returns **audio device**, **audio buffer**, **video buffer**, **data buffer**, **none**, **right**, **all**, or **left** for the destination of the data in the channel specified by **number**.

### LENGTH

Returns MCIERR\_INDETERMINATE\_LENGTH. The length of a CD-XA file cannot be determined in this release of OS/2 multimedia.

### MEDIA PRESENT

Returns TRUE if the media is inserted in the device; otherwise, it returns FALSE.

### MODE

Returns the current mode of the device: **not ready**, **stopped**, **playing**, **seeking**, **recording**, **paused**, or **other**.

### POSITION

Returns the current position.

### READY

Returns TRUE if the device is ready.

### TIME FORMAT

Returns the time format.

### VOLUME

Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

---

## STATUS (CD/XA Command) - Syntax Diagram

STATUS	object	CHANNEL number	
		LENGTH	WAIT
		MEDIA PRESENT	NOTIFY
		MODE	
		POSITION	
		READY	
		TIME FORMAT	
		VOLUME	

Examples

---

## STATUS (CD/XA Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## Digital Video Commands

The digital video device supports the following device-type specific commands and extensions to the following basic and required commands:

- [CAPABILITY](#)
- [CONNECTOR](#)
- [COPY](#)
- [CUE](#)
- [CUT](#)
- [DELETE](#)
- [INFO](#)
- [LOAD](#)
- [OPEN](#)
- [PASTE](#)
- [PLAY](#)
- [PUT](#)
- [RECORD](#)
- [REDO](#)
- [SAVE](#)
- [REWIND](#)
- [SEEK](#)
- [SET](#)
- [SETTUNER](#)
- [STATUS](#)
- [STEP](#)
- [UNDO](#)
- [WHERE](#)
- [WINDOW](#)

# CAPABILITY

---

## CAPABILITY (Digital Video Command) - Example

```
capability digitalvideo can distort wait
```

---

## CAPABILITY (Digital Video Command) - Purpose

The CAPABILITY command requests information about the capabilities of the device driver.

---

## CAPABILITY (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CAPABILITY (Digital Video Command) Keyword - CAN DISTORT

### **CAN DISTORT**

Returns FALSE for most frame-grabber types of hardware. However, some hardware, such as Video Blaster, is capable of performing independent scaling in the horizontal and vertical directions and return TRUE.

---

## CAPABILITY (Digital Video Command) Keyword - CAN EJECT

### **CAN EJECT**

Returns FALSE. The digital video device cannot eject media.

---

## CAPABILITY (Digital Video Command) Keyword - CAN LOCKEJECT

### **CAN LOCKEJECT**

Returns TRUE if the device can disable manual ejection of the media.

---

## CAPABILITY (Digital Video Command) Keyword - CAN PLAY

### **CAN PLAY**

Returns TRUE if the device can play.

---

## CAPABILITY (Digital Video Command) Keyword - CAN PROCESS INTERNAL

### **CAN PROCESS INTERNAL**

Returns TRUE if the device can internally process digital data with an internal digital to analog converter (DAC).

---

## CAPABILITY (Digital Video Command) Keyword - CAN RECORD

### **CAN RECORD**

Returns TRUE if the digital video device can record.

---

## CAPABILITY (Digital Video Command) Keyword - CAN RECORD INSERT

### **CAN RECORD INSERT**

Returns TRUE if the digital video device can insert data in the file. It expands the file and does not overwrite the information that is present.

---

## CAPABILITY (Digital Video Command) Keyword - CAN REVERSE

### **CAN REVERSE**

Returns FALSE. The digital video device can not play in reverse.

---

## CAPABILITY (Digital Video Command) Keyword - CAN SETVOLUME

### **CAN SETVOLUME**

Returns TRUE if the device supports software control of volume level.

---

## CAPABILITY (Digital Video Command) Keyword - CAN STREAM

### **CAN STREAM**

Returns TRUE if the device can continuously transfer digital data to and from another device. The source or destination of the data is determined by the device context connection.

---

## CAPABILITY (Digital Video Command) Keyword - CAN STRETCH

### **CAN STRETCH**

Returns FALSE for most frame-grabber types hardware. However, some hardware, such as Video Blaster, is capable of performing scaling and returns TRUE.

---

## CAPABILITY (Digital Video Command) Keyword - COMPOUND DEVICE

### **COMPOUND DEVICE**

Returns TRUE if the device requires an element name.

---

## CAPABILITY (Digital Video Command) Keyword - DEVICE TYPE

**DEVICE TYPE**  
Returns **Digitalvideo**.

---

## CAPABILITY (Digital Video Command) Keyword - FAST PLAY RATE

**FAST PLAY RATE**  
Returns twice the normal recorded playback rate. Returns 0 if the device cannot play fast.

---

## CAPABILITY (Digital Video Command) Keyword - HAS AUDIO

**HAS AUDIO**  
Returns TRUE if the device has audio playback.

---

## CAPABILITY (Digital Video Command) Keyword - HAS IMAGE

**HAS IMAGE**  
Returns FALSE. The digital video device does not support still image functions.

---

## CAPABILITY (Digital Video Command) Keyword - HAS TUNER

**HAS TUNER**  
Returns TRUE if the device has TV tuner capabilities.

---

## CAPABILITY (Digital Video Command) Keyword - HAS VIDEO

**HAS VIDEO**  
Returns TRUE.

---

## CAPABILITY (Digital Video Command) Keyword - HORIZONTAL IMAGE EXTENT

**HORIZONTAL IMAGE EXTENT**  
Returns the horizontal (X) source extent for images.

---

## CAPABILITY (Digital Video Command) Keyword - HORIZONTAL VIDEO EXTENT

**HORIZONTAL VIDEO EXTENT**  
Returns the horizontal (X) source extent for the video source.

---

## CAPABILITY (Digital Video Command) Keyword - MAXIMUM PLAY RATE

**MAXIMUM PLAY RATE**  
Not supported.

---

## CAPABILITY (Digital Video Command) Keyword - MESSAGE command

**MESSAGE command**  
Returns TRUE if the device supports the command specified by **command**. The **command** can be any string command such as

OPEN, PLAY, and so on.

---

## CAPABILITY (Digital Video Command) Keyword - MINIMUM PLAY RATE

**MINIMUM PLAY RATE**  
Not supported.

---

## CAPABILITY (Digital Video Command) Keyword - NORMAL PLAY RATE

**NORMAL PLAY RATE**  
Returns the normal recorded playback rate of the currently loaded motion video device element, in the current speed format, either as a percentage or in frames per second. Otherwise, returns 0.

---

## CAPABILITY (Digital Video Command) Keyword - OVERLAY GRAPHICS

**OVERLAY GRAPHICS**  
Returns FALSE. Overlay cards such as Video Blaster enable graphics overlay of the hardware monitor window, however, overlay is not supported over digital video playback in the graphics buffer.

---

## CAPABILITY (Digital Video Command) Keyword - PREROLL TIME

**PREROLL TIME**  
Returns 0, indicating the preroll time is not bounded.

---

## CAPABILITY (Digital Video Command) Keyword - PREROLL TYPE



**PREROLL TYPE**

Returns the preroll characteristics of the device. Returns NOTIFIED.

-----

## CAPABILITY (Digital Video Command) Keyword - SLOW PLAY RATE

**SLOW PLAY RATE**

Returns half the normal recorded playback rate. Returns 0 if the device cannot play slow.

-----

## CAPABILITY (Digital Video Command) Keyword - USES FILES

**USES FILES**

Returns TRUE if the element of a compound device is a file path name.

-----

## CAPABILITY (Digital Video Command) Keyword - VERTICAL IMAGE EXTENT

**VERTICAL IMAGE EXTENT**

Returns the vertical (Y) source extent for images.

-----

## CAPABILITY (Digital Video Command) Keyword - VERTICAL VIDEO EXTENT

**VERTICAL VIDEO EXTENT**

Returns the vertical (Y) source extent for the video source.

-----

## CAPABILITY (Digital Video Command) Keyword - WINDOWS

**WINDOWS**

Not supported.

---

## CAPABILITY (Digital Video Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

---

## CAPABILITY (Digital Video Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returns immediately to the application. When the requested action is complete an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CAPABILITY (Digital Video Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### **CAN DISTORT**

Returns FALSE for most frame-grabber types of hardware. However, some hardware, such as Video Blaster, is capable of performing independent scaling in the horizontal and vertical directions and return TRUE.

### **CAN EJECT**

Returns FALSE. The digital video device cannot eject media.

### **CAN LOCKEJECT**

Returns TRUE if the device can disable manual ejection of the media.

### **CAN PLAY**

Returns TRUE if the device can play.

### **CAN PROCESS INTERNAL**

Returns TRUE if the device can internally process digital data with an internal digital to analog converter (DAC).

### **CAN RECORD**

Returns TRUE if the digital video device can record.

### **CAN RECORD INSERT**

Returns TRUE if the digital video device can insert data in the file. It expands the file and does not overwrite the information that is present.

### **CAN REVERSE**

Returns FALSE. The digital video device can not play in reverse.

### **CAN SETVOLUME**

Returns TRUE if the device supports software control of volume level.

### **CAN STREAM**

Returns TRUE if the device can continuously transfer digital data to and from another device. The source or destination of the data is determined by the device context connection.

#### **CAN STRETCH**

Returns FALSE for most frame-grabber types hardware. However, some hardware, such as Video Blaster, is capable of performing scaling and returns TRUE.

#### **COMPOUND DEVICE**

Returns TRUE if the device requires an element name.

#### **DEVICE TYPE**

Returns **Digitalvideo**.

#### **FAST PLAY RATE**

Returns twice the normal recorded playback rate. Returns 0 if the device cannot play fast.

#### **HAS AUDIO**

Returns TRUE if the device has audio playback.

#### **HAS IMAGE**

Returns FALSE. The digital video device does not support still image functions.

#### **HAS TUNER**

Returns TRUE if the device has TV tuner capabilities.

#### **HAS VIDEO**

Returns TRUE.

#### **HORIZONTAL IMAGE EXTENT**

Returns the horizontal (X) source extent for images.

#### **HORIZONTAL VIDEO EXTENT**

Returns the horizontal (X) source extent for the video source.

#### **MAXIMUM PLAY RATE**

Not supported.

#### **MESSAGE command**

Returns TRUE if the device supports the command specified by **command**. The **command** can be any string command such as OPEN, PLAY, and so on.

#### **MINIMUM PLAY RATE**

Not supported.

#### **NORMAL PLAY RATE**

Returns the normal recorded playback rate of the currently loaded motion video device element, in the current speed format, either as a percentage or in frames per second. Otherwise, returns 0.

#### **OVERLAY GRAPHICS**

Returns FALSE. Overlay cards such as Video Blaster enable graphics overlay of the hardware monitor window, however, overlay is not supported over digital video playback in the graphics buffer.

#### **PREROLL TIME**

Returns 0, indicating the preroll time is not bounded.

#### **PREROLL TYPE**

Returns the preroll characteristics of the device. Returns NOTIFIED.

#### **SLOW PLAY RATE**

Returns half the normal recorded playback rate. Returns 0 if the device cannot play slow.

#### **USES FILES**

Returns TRUE if the element of a compound device is a file path name.

#### **VERTICAL IMAGE EXTENT**

Returns the vertical (Y) source extent for images.

#### **VERTICAL VIDEO EXTENT**

Returns the vertical (Y) source extent for the video source.

#### **WINDOWS**

Not supported.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

**NOTIFY**

The command is executed asynchronously, allowing control to be returns immediately to the application. When the requested action is complete an MM\_MCINOTIFY message is sent to the application window procedure.

CAPABILITY (Digital Video Command) - Syntax Diagram

CAPABILITY	object	CAN DISTORT	
		CAN EJECT	WAIT
		CAN LOCKEJECT	NOTIFY
		CAN PLAY	
		CAN PROCESS INTERNAL	
		CAN RECORD	
		CAN RECORD INSERT	
		CAN REVERSE	
		CAN SETVOLUME	
		CAN STREAM	
		CAN STRETCH	
		COMPOUND DEVICE	
		DEVICE TYPE	
		FAST PLAY RATE	
		HAS AUDIO	
		HAS IMAGE	
		HAS TUNER	
		HAS VIDEO	
		HORIZONTAL VIDEO EXTENT	
		HORIZONTAL IMAGE EXTENT	
		MAXIMUM PLAY RATE	
		MESSAGE command	
		MINIMUM PLAY RATE	
		NORMAL PLAY RATE	
		OVERLAY GRAPHICS	
		PREROLL TIME	
		PREROLL TYPE	
		SLOW PLAY RATE	
		USES FILES	
		VERTICAL IMAGE EXTENT	
		VERTICAL VIDEO EXTENT	
		WINDOWS	



CAPABILITY (Digital Video Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

# CAPTURE

---

## CAPTURE (Digital Video Command) - Example

```
capture digitalvideo at 100 100 260 220 convert wait
```

---

## CAPTURE (Digital Video Command) - Purpose

The CAPTURE command captures the current video image. This does not cause the image or bitmap to be saved; the application must subsequently issue a SAVE command to save the device element. The device will freeze motion temporarily if needed to capture the image and put the image into the image device element. Repeated capture operations will overwrite the image contained in this temporary space. The device will wait for a [SAVE](#) command to transfer the information to a file. Use [MCI\\_GETIMAGEBUFFER](#) to supply an application with a copy of the capture video buffer.

**Note:** If no rectangle is specified, the entire rectangle is captured.

---

## CAPTURE (Digital Video Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CAPTURE (Digital Video Command) Keyword - AT rect

### AT rect

Specifies a rectangle relative to the window origin in device coordinates. The rectangle is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner. Only the video in that subregion will be captured.

---

## CAPTURE (Digital Video Command) Keyword - CONVERT

**CONVERT**

Specifies that the image format will be converted to the OS/2 bitmap format. The default is the device-specific format.

---

## CAPTURE (Digital Video Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## CAPTURE (Digital Video Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CAPTURE (Digital Video Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**AT rect**

Specifies a rectangle relative to the window origin in device coordinates. The rectangle is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner. Only the video in that subregion will be captured.

**CONVERT**

Specifies that the image format will be converted to the OS/2 bitmap format. The default is the device-specific format.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CAPTURE (Digital Video Command) - Syntax Diagram

CAPTURE

object

AT rect  
CONVERT

WAIT  
NOTIFY

Examples

---

## CAPTURE (Digital Video Command) - Topics

Select an item:

[Purpose](#)

[Syntax Diagram](#)

[Keywords](#)

[Example](#)

[Glossary](#)

---

## CONNECTOR

---

## CONNECTOR (Digital Video Command) - Example

This enables video input connector number 2 on the capture adapter.

```
connector digitalvideo enable type video in number 2 wait
```

---

## CONNECTOR (Digital Video Command) - Purpose

The CONNECTOR command enables, disables, or queries the status of connector on a device.

---

## CONNECTOR (Digital Video Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name

- Filename
- Alias

-----

## CONNECTOR (Digital Video Command) Keyword - ENABLE

### **ENABLE**

Enables information flow through the indicated connector. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

-----

## CONNECTOR (Digital Video Command) Keyword - DISABLE

### **DISABLE**

Disables information flow through the indicated connector. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

-----

## CONNECTOR (Digital Video Command) Keyword - QUERY

### **QUERY**

Queries the state of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

-----

## CONNECTOR (Digital Video Command) Keyword - NUMBER connector\_number

### **NUMBER connector\_number**

Indicates the connector number on which to perform the requested action. If this item is omitted, then the first connector is assumed. If the TYPE item is included, then the connector number is interpreted as a relative offset within the specified connector type.

-----

## CONNECTOR (Digital Video Command) Keyword - TYPE connector\_type

### **TYPE connector\_type**

Indicates the type of connector to which the requested action applies. The connector types are defined by each device. Valid connector type values for the digital video device include:



- line in
- video in

-----

## CONNECTOR (Digital Video Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## CONNECTOR (Digital Video Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CONNECTOR (Digital Video Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### **ENABLE**

Enables information flow through the indicated connector. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

### **DISABLE**

Disables information flow through the indicated connector. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

### **QUERY**

Queries the state of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

### **NUMBER connector\_number**

Indicates the connector number on which to perform the requested action. If this item is omitted, then the first connector is assumed. If the TYPE item is included, then the connector number is interpreted as a relative offset within the specified connector type.

### **TYPE connector\_type**

Indicates the type of connector to which the requested action applies. The connector types are defined by each device. Valid connector type values for the digital video device include:

- line in
- video in

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the

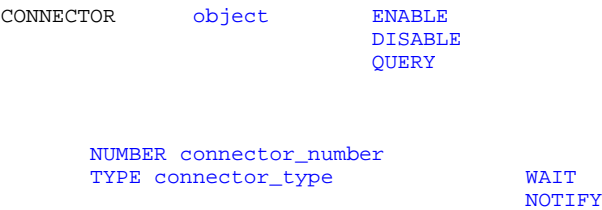
application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CONNECTOR (Digital Video Command) - Syntax Diagram



---

## CONNECTOR (Digital Video Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

## COPY

---

## COPY (Digital Video Command) - Example

```
copy digitalvideo from 10000 to 50000
```

---

## COPY (Digital Video Command) - Purpose

The COPY command copies information from a file into the clipboard.

---

## COPY (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## COPY (Digital Video Command) Keyword - FROM pos

### **FROM pos**

The position to start copying. If FROM is omitted, the copy starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

---

## COPY (Digital Video Command) Keyword - TO pos

### **TO pos**

The position to stop copying. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

---

## COPY (Digital Video Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## COPY (Digital Video Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## COPY (Digital Video Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### FROM pos

The position to start copying. If FROM is omitted, the copy starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

### TO pos

The position to stop copying. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## COPY (Digital Video Command) - Syntax Diagram

COPY      [object](#)      [FROM pos](#)      [WAIT](#)  
                                 [TO pos](#)      [NOTIFY](#)

Examples

---

## COPY (Digital Video Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

# CUE

---

## CUE (Digital Video Command) - Example

The following command prepares the device for recording.

```
cue digitalvideo input wait
```

The following command prepares the device for playback and positions the media at the specified location, without displaying a video window.

```
cue digitalvideo to 70 wait
```

---

## CUE (Digital Video Command) - Purpose

The CUE command prepares for playback or recording. The CUE command does not have to be issued prior to playback or recording. However, depending on the device, it can reduce the delay associated with the [PLAY](#) or [RECORD](#) command.

The CUE command is not related to the [SETCUEPOINT](#) command.

---

## CUE (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CUE (Digital Video Command) Keyword - INPUT

### **INPUT**

Prepares the device for recording.

---

## CUE (Digital Video Command) Keyword - OUTPUT

### **OUTPUT**

Prepares the device for playback. This is the default.

---

## CUE (Digital Video Command) Keyword - NOSHOW

### **NOSHOW**

Causes the window to be hidden while the cue operation is performed. This is the default.

---

## CUE (Digital Video Command) Keyword - SHOW

### **SHOW**

Causes the window to be displayed while the cue operation is performed.

---

## CUE (Digital Video Command) Keyword - TO pos

### **TO pos**

Specifies a position to seek to when cueing the device for playback. If the TO position is beyond the end of the media or segment, an MCIERR\_OUTOFRANGE error is returned. The default TO position is 0 (the beginning of the media).

If the [STATUS](#) command is issued following the CUE command to retrieve the current position, the position returned will indicate the next frame.

---

## CUE (Digital Video Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## CUE (Digital Video Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# CUE (Digital Video Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**INPUT**

Prepares the device for recording.

**OUTPUT**

Prepares the device for playback. This is the default.

**NOSHOW**

Causes the window to be hidden while the cue operation is performed. This is the default.

**SHOW**

Causes the window to be displayed while the cue operation is performed.

**TO pos**

Specifies a position to seek to when cueing the device for playback. If the TO position is beyond the end of the media or segment, an MCIERR\_OUTOFRANGE error is returned. The default TO position is 0 (the beginning of the media).

If the [STATUS](#) command is issued following the CUE command to retrieve the current position, the position returned will indicate the next frame.

**WAIT**

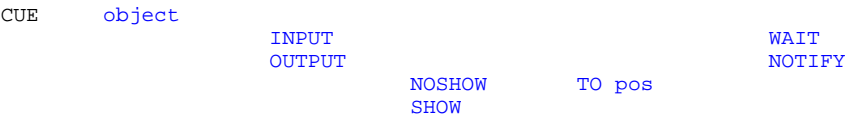
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# CUE (Digital Video Command) - Syntax Diagram



Examples

-----

# CUE (Digital Video Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## CUT

---

## CUT (Digital Video Command) - Example

```
cut digitalvideo from 10000 to 40000 wait
```

---

## CUT (Digital Video Command) - Purpose

The CUT command cuts removes the specified range and places the data in the clipboard.

---

## CUT (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CUT (Digital Video Command) Keyword - FROM pos

### **FROM pos**

The position to start cutting. If FROM is omitted, the cut starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.



---

## CUT (Digital Video Command) Keyword - TO pos

### TO pos

The position to stop cutting. If FROM is omitted, the cut starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

---

## CUT (Digital Video Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## CUT (Digital Video Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CUT (Digital Video Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### FROM pos

The position to start cutting. If FROM is omitted, the cut starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

### TO pos

The position to stop cutting. If FROM is omitted, the cut starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CUT (Digital Video Command) - Syntax Diagram

CUT      *object*

                 FROM pos      WAIT

                 TO pos      NOTIFY

Examples

---

## CUT (Digital Video Command) - Topics

Select an item:

- [Purpose](#)
- [Syntax Diagram](#)
- [Keywords](#)
- [Example](#)
- [Glossary](#)

---

## DELETE

---

## DELETE (Digital Video Command) - Example

```
delete digitalvideo from 10000 to 40000 wait
```

---

## DELETE (Digital Video Command) - Purpose

The DELETE command deletes information from a file.

---

## DELETE (Digital Video Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## DELETE (Digital Video Command) Keyword - FROM pos

### FROM pos

The position to start deleting. If FROM is omitted, the delete starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

-----

## DELETE (Digital Video Command) Keyword - TO pos

### TO pos

The position to stop deleting. If FROM is omitted, the delete starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

-----

## DELETE (Digital Video Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## DELETE (Digital Video Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# DELETE (Digital Video Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

The units of the TO and FROM positions must be supplied in the currently selected time format.

## FROM pos

The position to start deleting. If FROM is omitted, the delete starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

## TO pos

The position to stop deleting. If FROM is omitted, the delete starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

## WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# DELETE (Digital Video Command) - Syntax Diagram

DELETE      object

                 FROM pos      WAIT

                 TO pos      NOTIFY



-----

# DELETE (Digital Video Command) - Topics

Select an item:

- [Purpose](#)
- [Syntax Diagram](#)
- [Keywords](#)
- [Example](#)
- [Glossary](#)

---

## INFO

---

### INFO (Digital Video Command) - Example

```
info digitalvideo product wait
```

---

### INFO (Digital Video Command) - Purpose

The INFO command fills a user-supplied buffer with information.

---

### INFO (Digital Video Command) Keyword - object

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

### INFO (Digital Video Command) Keyword - FILE

#### **FILE**

Returns the filename of the digital video file.

---

### INFO (Digital Video Command) Keyword - PRODUCT

#### **PRODUCT**

Returns the product name and model of the current digital video device.

-----

## INFO (Digital Video Command) Keyword - REGION

### **REGION**

Returns the name of the current tuner region.

-----

## INFO (Digital Video Command) Keyword - REGION TEXT

### **REGION TEXT**

Returns a description of the current tuner region.

-----

## INFO (Digital Video Command) Keyword - WINDOW TEXT

### **WINDOW TEXT**

Returns the caption of the display window.

-----

## INFO (Digital Video Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## INFO (Digital Video Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## INFO (Digital Video Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**FILE**

Returns the filename of the digital video file.

**PRODUCT**

Returns the product name and model of the current digital video device.

**REGION**

Returns the name of the current tuner region.

**REGION TEXT**

Returns a description of the current tuner region.

**WINDOW TEXT**

Returns the caption of the display window.

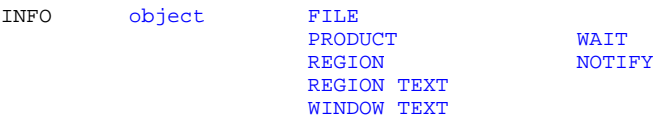
**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

INFO (Digital Video Command) - Syntax Diagram



INFO (Digital Video Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

## LOAD (Digital Video Command) - Example

```
load digitalvideo movie.avi wait
```

---

## LOAD (Digital Video Command) - Purpose

The LOAD command loads a new device element (file) into an already open device context.

---

## LOAD (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## LOAD (Digital Video Command) Keyword - filename

### **filename**

Specifies the name of the file to load.

---

## LOAD (Digital Video Command) Keyword - NEW

### **NEW**

Creates a temporary element for subsequent use with the [RECORD](#) command (removing any previous cue input). The temporary file can be made permanent by providing a name using the [SAVE](#) command.

---

## LOAD (Digital Video Command) Keyword - READONLY



**READONLY**

Opens the file in a read-only mode and prevents inadvertent changes to the file.

---

## LOAD (Digital Video Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## LOAD (Digital Video Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## LOAD (Digital Video Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**filename**

Specifies the name of the file to load.

**NEW**

Creates a temporary element for subsequent use with the [RECORD](#) command (removing any previous cue input). The temporary file can be made permanent by providing a name using the [SAVE](#) command.

**READONLY**

Opens the file in a read-only mode and prevents inadvertent changes to the file.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## LOAD (Digital Video Command) - Syntax Diagram

LOAD      object      filename      WAIT  
NEW      READONLY      NOTIFY

Examples

## LOAD (Digital Video Command) - Topics

Select an item:  
[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

## OPEN

## OPEN (Digital Video Command) - Example

```
open digitalvideo alias w shareable wait
```

## OPEN (Digital Video Command) - Purpose

The OPEN command initializes the device.

## OPEN (Digital Video Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

## OPEN (Digital Video Command) Keyword - ALIAS device\_alias

**ALIAS device\_alias**

Specifies an alternate name for the device. If specified, it must also be used for subsequent references.

---

## OPEN (Digital Video Command) Keyword - DOSQUEUE

**DOSQUEUE**

If a device instance is opened with the DOSQUEUE keyword specified, window handles that are passed in for the instance will be treated as OS/2 Control Program queue handles.

---

## OPEN (Digital Video Command) Keyword - PARENT hwnd

**PARENT hwnd**

Specifies the window handle of the parent window as a character representation of the decimal window handle value. If specified, it is used as the parent window of the digital video device default window.

---

## OPEN (Digital Video Command) Keyword - READONLY

**READONLY**

Specifies that the file is to be opened in read-only mode.

---

## OPEN (Digital Video Command) Keyword - SHAREABLE

**SHAREABLE**

Initializes the device as shareable. Specifying SHAREABLE makes the resources of the device available to other device contexts. If

SHAREABLE is not specified on OPEN, the resource will be exclusively acquired when the device is opened.

---

## OPEN (Digital Video Command) Keyword - TYPE device\_type

### **TYPE device\_type**

Specifies the compound device used to control a device element. As an alternative to TYPE, the media control interface can use the .TYPE extended attribute or file extension associated with the file to select the controlling device.

---

## OPEN (Digital Video Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## OPEN (Digital Video Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## OPEN (Digital Video Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### **ALIAS device\_alias**

Specifies an alternate name for the device. If specified, it must also be used for subsequent references.

### **DOSQUEUE**

If a device instance is opened with the DOSQUEUE keyword specified, window handles that are passed in for the instance will be treated as OS/2 Control Program queue handles.

### **PARENT hwnd**

Specifies the window handle of the parent window as a character representation of the decimal window handle value. If specified, it is used as the parent window of the digital video device default window.

### **READONLY**

Specifies that the file is to be opened in read-only mode.

**SHAREABLE**

Initializes the device as shareable. Specifying SHAREABLE makes the resources of the device available to other device contexts. If SHAREABLE is not specified on OPEN, the resource will be exclusively acquired when the device is opened.

**TYPE device\_type**

Specifies the compound device used to control a device element. As an alternative to TYPE, the media control interface can use the .TYPE extended attribute or file extension associated with the file to select the controlling device.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

OPEN (Digital Video Command) - Syntax Diagram

OPEN

object

ALIAS device\_alias  
DOSQUEUE  
PARENT hwnd  
READONLY  
SHAREABLE  
TYPE device\_type

WAIT  
NOTIFY



OPEN (Digital Video Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

PASTE

PASTE (Digital Video Command) - Example

```
open digitalvideo alias video1 shareable wait
load video1 movie.avi wait
paste video1 from 10000 to 40000 wait
```

---

## PASTE (Digital Video Command) - Purpose

The PASTE command pastes information from the clipboard into a file. The media position after a paste operation is at the end of the pasted data.

---

## PASTE (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

## PASTE (Digital Video Command) Keyword - FROM pos

### **FROM pos**

The position to start pasting. If FROM is omitted, the paste starts at the current position.

---

## PASTE (Digital Video Command) Keyword - TO pos

### **TO pos**

The position to stop pasting. The pasted data *replaces* the data from the FROM position (or the current position if FROM is not specified) to the TO position.

If TO is omitted, the end of file is assumed and the pasted data is *inserted* starting at the FROM position (or the current position if FROM is not specified).

---

## PASTE (Digital Video Command) Keyword - WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

-----

- Device type
- Device name
- Filename
- Alias

If TO is omitted, the end of file is assumed and the pasted data is *inserted* starting at the FROM position (or the current position if FROM is not specified).

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

---

```
PASTE      object
           FROM pos
           TO pos
           WAIT
           NOTIFY
```

## Examples

---

# PASTE (Digital Video Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## PLAY

---

# PLAY (Digital Video Command) - Example

```
play digitalvideo notify
```

---

# PLAY (Digital Video Command) - Purpose

The PLAY command starts a play operation on the device.

---

# PLAY (Digital Video Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

# PLAY (Digital Video Command) Keyword - FROM pos



**FROM pos**

Specifies the frame at which to start playing. If FROM is omitted, PLAY starts at the current position.

---

## PLAY (Digital Video Command) Keyword - TO pos

**TO pos**

Specifies the frame at which to stop playing. If TO is omitted, PLAY stops at the end frame.

---

## PLAY (Digital Video Command) Keyword - FAST

**FAST**

Plays the digital video sequence at twice the normal recorded playback rate (no audio).

---

## PLAY (Digital Video Command) Keyword - SLOW

**SLOW**

Plays the digital video sequence at half the normal recorded playback rate (no audio).

---

## PLAY (Digital Video Command) Keyword - REVERSE

**REVERSE**

Indicates the play direction for the device is backwards. You cannot use the TO keyword with REVERSE.

---

## PLAY (Digital Video Command) Keyword - SCAN

**SCAN**

Plays frames when indexed. Otherwise, plays the digital video as fast as possible without disabling video (no audio). You cannot use the TO keyword with SCAN.

---

## PLAY (Digital Video Command) Keyword - SPEED units

**SPEED units**

Plays the digital video sequence at the specified speed. Speed is specified in units specified by **set speed format**. (See the [SET](#) command.)

---

## PLAY (Digital Video Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## PLAY (Digital Video Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## PLAY (Digital Video Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**FROM pos**

Specifies the frame at which to start playing. If FROM is omitted, PLAY starts at the current position.

**TO pos**

Specifies the frame at which to stop playing. If TO is omitted, PLAY stops at the end frame.

**FAST**

Plays the digital video sequence at twice the normal recorded playback rate (no audio).

**SLOW**

Plays the digital video sequence at half the normal recorded playback rate (no audio).

**REVERSE**

Indicates the play direction for the device is backwards. You cannot use the TO keyword with REVERSE.

**SCAN**

Plays frames when indexed. Otherwise, plays the digital video as fast as possible without disabling video (no audio). You cannot use the TO keyword with SCAN.

**SPEED units**

Plays the digital video sequence at the specified speed. Speed is specified in units specified by **set speed format**. (See the [SET](#) command.)

**WAIT**

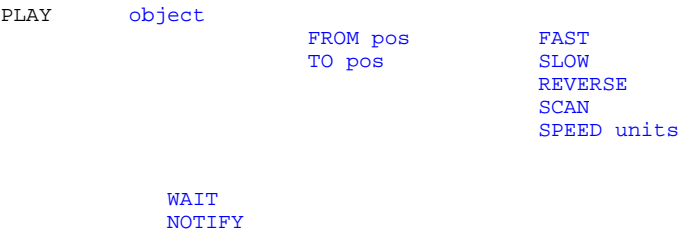
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## PLAY (Digital Video Command) - Syntax Diagram



---

## PLAY (Digital Video Command) - Remarks

An MCIERR\_OUTOFRANGE error is returned if the range specified with the FROM and TO keywords is not large enough.

If you are using an application-defined window and your application is running on a system without direct-access device driver support for motion video, do *not* specify WAIT with the PLAY command unless the thread issuing the message is separate from the thread reading the message queue.

---

## PLAY (Digital Video Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Remarks](#)
  - [Example](#)
  - [Glossary](#)

---

## PUT

---

## PUT (Digital Video Command) - Example

The following command sets the monitor window position at 100, 100.

```
put digitalvideo window at 100 100 260 220 move monitor wait
```

The following command sets the video capture region to be a 640x288 rectangle at offset 0, 100 (from the lower left) out of the total video capture extents.

```
put digitalvideo record source at 0 100 640 388 move wait
```

The following command sets the output video size to 320x144.

```
put digitalvideo record destination at 0 0 320 144 wait
```

---

## PUT (Digital Video Command) - Purpose

The PUT command defines the source and destination windows.

---

## PUT (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## PUT (Digital Video Command) Keyword - DESTINATION AT rect

### **DESTINATION AT rect**

Determines the size and position of the playback video relative to the playback window. The rectangle is relative to the window origin in device coordinates and is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner.

---

# PUT (Digital Video Command) Keyword - WINDOW AT rect

## **WINDOW AT rect**

Specifies the size and position of the playback window (either a default window or an application-defined window).

**Note:** The MOVE and SIZE options can both be specified, in which case, the window is moved and sized at the same time.

---

# PUT (Digital Video Command) Keyword - MOVE

## **MOVE**

Moves the appropriate window to the X1 Y1 coordinates specified in the rectangle. Window coordinates are relative to the parent window.

All coordinates of the rectangle (X1 Y1 X2 Y2) must be specified, but X2 Y2 are ignored if the SIZE keyword is not specified.

This option will not affect an application-supplied alternate video window.

---

# PUT (Digital Video Command) Keyword - SIZE

## **SIZE**

Sizes the appropriate window to be (X2 - X1) and (Y2 - Y1). All coordinates of the rectangle (X1 Y1 X2 Y2) must be specified.

This option will not affect an application-supplied alternate video window.

---

# PUT (Digital Video Command) Keyword - MONITOR

## **MONITOR**

Specifies the position and size of the window containing the monitor rectangle.

**Note:** The MOVE and SIZE options can both be specified, in which case, the monitor video window is moved and sized at the same time.

---

# PUT (Digital Video Command) Keyword - RECORD DESTINATION AT rect

## **RECORD DESTINATION AT rect**

Determines the size of the movie to be played back (playback video). The X1 and Y1 rectangle coordinates are subtracted from X2 and Y2, respectively, to determine the playback video size. For example, the following command:

PUT object RECORD DESTINATION AT 13 47 173 167

yields a playback video size of 160x120 ((173 - 13)x(167 - 47)).

If either the width or the height of the rectangle specified with the RECORD and DESTINATION keywords (indicating playback video size) is not a multiple of eight, that value is rounded to the nearest multiple of eight.

Use the **WHERE object RECORD DESTINATION ADJUSTED** command to obtain the actual size of the playback video.

---

## PUT (Digital Video Command) Keyword - RECORD SOURCE AT rect

### RECORD SOURCE AT rect

Specifies the origin and size of a window describing video to be captured relative to the maximum available capture window. The rectangle coordinates for source capture are relative to the lower-left corner of the video source.

If the device is cued for input (recording), the actual source rectangle is displayed. Otherwise, the maximum source rectangle is displayed with any subset represented as an animated, dashed-line rectangle.

**Note:** If only the source is set then the destination defaults to half of the source size. The video source extent can be found using the [STATUS](#) command with the HORIZONTAL VIDEO EXTENT and VERTICAL VIDEO EXTENT keywords.

If the device cannot distort and the rectangle specified with **PUT object RECORD SOURCE AT rect** is not an integral multiple of the rectangle specified with **PUT object RECORD DESTINATION AT rect** (playback video size), the source and destination rectangles will be adjusted to find the nearest values that will make the source become an integral multiple of the destination and the destination become a multiple of eight.

---

## PUT (Digital Video Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## PUT (Digital Video Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## PUT (Digital Video Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**DESTINATION AT rect**

Determines the size and position of the playback video relative to the playback window. The rectangle is relative to the window origin in device coordinates and is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner.

**WINDOW AT rect**

Specifies the size and position of the playback window (either a default window or an application-defined window).

**Note:** The MOVE and SIZE options can both be specified, in which case, the window is moved and sized at the same time.

**MOVE**

Moves the appropriate window to the X1 Y1 coordinates specified in the rectangle. Window coordinates are relative to the parent window.

All coordinates of the rectangle (X1 Y1 X2 Y2) must be specified, but X2 Y2 are ignored if the SIZE keyword is not specified.

This option will not affect an application-supplied alternate video window.

**SIZE**

Sizes the appropriate window to be (X2 - X1) and (Y2 - Y1). All coordinates of the rectangle (X1 Y1 X2 Y2) must be specified.

This option will not affect an application-supplied alternate video window.

**MONITOR**

Specifies the position and size of the window containing the monitor rectangle.

**Note:** The MOVE and SIZE options can both be specified, in which case, the monitor video window is moved and sized at the same time.

**RECORD DESTINATION AT rect**

Determines the size of the movie to be played back (playback video). The X1 and Y1 rectangle coordinates are subtracted from X2 and Y2, respectively, to determine the playback video size. For example, the following command:

```
PUT object RECORD DESTINATION AT 13 47 173 167
```

yields a playback video size of 160x120 ((173 - 13)x(167 - 47)).

If either the width or the height of the rectangle specified with the RECORD and DESTINATION keywords (indicating playback video size) is not a multiple of eight, that value is rounded to the nearest multiple of eight.

Use the **WHERE object RECORD DESTINATION ADJUSTED** command to obtain the actual size of the playback video.

**RECORD SOURCE AT rect**

Specifies the origin and size of a window describing video to be captured relative to the maximum available capture window. The rectangle coordinates for source capture are relative to the lower-left corner of the video source.

If the device is cued for input (recording), the actual source rectangle is displayed. Otherwise, the maximum source rectangle is displayed with any subset represented as an animated, dashed-line rectangle.

**Note:** If only the source is set then the destination defaults to half of the source size. The video source extent can be found using the [STATUS](#) command with the HORIZONTAL VIDEO EXTENT and VERTICAL VIDEO EXTENT keywords.

If the device cannot distort and the rectangle specified with **PUT object RECORD SOURCE AT rect** is not an integral multiple of the rectangle specified with **PUT object RECORD DESTINATION AT rect** (playback video size), the source and destination rectangles will be adjusted to find the nearest values that will make the source become an integral multiple of the destination and the destination become a multiple of eight.

**Note:** The source rectangle specifies the portion of the image to be captured and the destination rectangle specifies the size of the video to be recorded. This indicates the scaling to be applied to the source rectangle.

**WAIT**

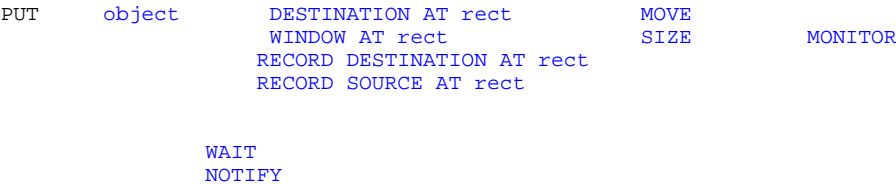
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## PUT (Digital Video Command) - Syntax Diagram



-----

## PUT (Digital Video Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

-----

## RECORD

-----

## RECORD (Digital Video Command) - Example

record digitalvideo to 1000 wait

-----



# RECORD (Digital Video Command) - Purpose

The RECORD command initiates real-time recording of motion video with audio capture.

The functionality of video record, capture, and monitor is performed through an media control interface logical device. An association between a logical device name and any video capture cards is established automatically during installation. *Digital/video01* is the logical device that supports playback only, with unmodified functionality from the original OS/2 multimedia installation, and is not associated with a capture device. *Digital/video02* (and higher) enables capture and recording, and is associated with capture hardware. These logical device names are produced by default through installation of OS/2 multimedia and Video IN for OS/2.

**Note:** Only a single actively recording instance is supported at a time, as video capture hardware does not support concurrent use. Multiple instances of the MCD can be opened, but only one recording instance can be open.

---

## RECORD (Digital Video Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

## RECORD (Digital Video Command) Keyword - TO pos

### TO pos

Specifies the position to stop recording. TO is used only as an indication of the length of the recording to be performed. A STOP is performed implicitly if the device is not stopped when RECORD is issued.

---

## RECORD (Digital Video Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## RECORD (Digital Video Command) Keyword - NOTIFY

### NOTIFY

-----

**object**

- Device type
- Device name
- Filename
- Alias

**WAIT**

**NOTIFY**

-----

```
RECORD      object
            TO pos
            WAIT
            NOTIFY
```



# REDO

---

## REDO (Digital Video Command) - Example

```
open macaw.avi alias vid shareable wait
cut vid to 6000 wait
undo vid wait
redo vid wait
```

---

## REDO (Digital Video Command) - Purpose

The REDO command redoes the last editing action (cut, paste, or delete) which was undone with the [UNDO](#) command. REDO should immediately follow [UNDO](#); otherwise, editing actions performed after [UNDO](#) (and before a corresponding REDO) will be lost when the REDO command is issued. The position of the media after a REDO is 0.

Multiple REDO operations are permitted, corresponding to the number of editing operations that have been previously undone with the [UNDO](#) command.

---

## REDO (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## REDO (Digital Video Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## REDO (Digital Video Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## REDO (Digital Video Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## REDO (Digital Video Command) - Syntax Diagram

REDO      [object](#)

[WAIT](#)  
[NOTIFY](#)



-----

## REDO (Digital Video Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

-----

## REWIND

---

## REWIND (Digital Video Command) - Example

```
rewind digitalvideo wait
```

---

## REWIND (Digital Video Command) - Purpose

The REWIND command rewinds or seeks the device element to the first playable position (beginning).

---

## REWIND (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## REWIND (Digital Video Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## REWIND (Digital Video Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## REWIND (Digital Video Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

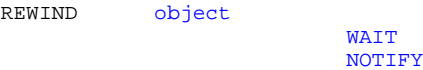
**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

REWIND (Digital Video Command) - Syntax Diagram



REWIND (Digital Video Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

SAVE

SAVE (Digital Video Command) - Example

```
save video1 mymovie.avi wait
```

---

## SAVE (Digital Video Command) - Purpose

The SAVE command saves the current file.

---

## SAVE (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## SAVE (Digital Video Command) Keyword - filename

### **filename**

Specifies the destination path and file name.

---

## SAVE (Digital Video Command) Keyword - VIDEO

### **VIDEO**

The file to be saved is a video file. This is the default.

---

## SAVE (Digital Video Command) Keyword - IMAGE

### **IMAGE**

The file to be saved is a still image file.

---

## SAVE (Digital Video Command) Keyword - WAIT

### **WAIT**

-----

-----

-----





---

## SAVE (Digital Video Command) - Topics

Select an item:

[Purpose](#)

[Syntax Diagram](#)

[Keywords](#)

[Example](#)

[Glossary](#)

---

## SEEK

---

## SEEK (Digital Video Command) - Example

```
seek digitalvideo to start wait
```

---

## SEEK (Digital Video Command) - Purpose

The SEEK command seeks finds the specified position in the file.

---

## SEEK (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

## SEEK (Digital Video Command) Keyword - TO pos

#### **TO pos**

Specifies the final position for the seek.

---

## SEEK (Digital Video Command) Keyword - TO NEAREST pos

#### **TO NEAREST pos**

Seeks to the nearest I-frame preceding the point specified by **pos**.

---

## SEEK (Digital Video Command) Keyword - TO START

#### **TO START**

Seeks to the start of the file.

---

## SEEK (Digital Video Command) Keyword - TO END

#### **TO END**

Seeks to the end of the file.

---

## SEEK (Digital Video Command) Keyword - WAIT

#### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## SEEK (Digital Video Command) Keyword - NOTIFY

#### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## SEEK (Digital Video Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**TO pos**

Specifies the final position for the seek.

**TO NEAREST pos**

Seeks to the nearest I-frame preceding the point specified by **pos**.

**TO START**

Seeks to the start of the file.

**TO END**

Seeks to the end of the file.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# SEEK (Digital Video Command) - Syntax Diagram

SEEK	<a href="#">object</a>	<a href="#">TO pos</a>	
		<a href="#">TO NEAREST pos</a>	<a href="#">WAIT</a>
		<a href="#">TO START</a>	<a href="#">NOTIFY</a>
		<a href="#">TO END</a>	



-----

# SEEK (Digital Video Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

-----

# SET

---

## SET (Digital Video Command) - Example

```
set digitalvideo time format milliseconds wait
```

---

## SET (Digital Video Command) - Purpose

The SET command sets the various control and attribute items.

---

## SET (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## SET (Digital Video Command) Keyword - AUDIO

### **AUDIO**

Specifies the audio attributes of the device context determined by the ALL, LEFT, RIGHT, OVER, and VOLUME keywords.

---

## SET (Digital Video Command) Keyword - ALL

### **ALL**

Applies to both or all of the channels (default).

Specify ON or OFF with the ALL keyword.

---

## SET (Digital Video Command) Keyword - ON

**ON**  
Enables audio.

-----

## SET (Digital Video Command) Keyword - OFF

**OFF**  
Disables audio.

-----

## SET (Digital Video Command) Keyword - LEFT

**LEFT**  
Applies to the left channel.  
Specify ON or OFF with the LEFT keyword.

-----

## SET (Digital Video Command) Keyword - ON

**ON**  
Enables audio for the left channel.

-----

## SET (Digital Video Command) Keyword - OFF

**OFF**  
Disables audio for the left channel.

-----

## SET (Digital Video Command) Keyword - RIGHT

**RIGHT**  
Applies to the right channel.  
Specify ON or OFF with the RIGHT keyword.

---

## SET (Digital Video Command) Keyword - ON

**ON**  
Enables audio for the right channel.

---

## SET (Digital Video Command) Keyword - OFF

**OFF**  
Disables audio for the right channel.

---

## SET (Digital Video Command) Keyword - OVER milliseconds

**OVER milliseconds**  
Applies the change over the specified time period (fade).

---

## SET (Digital Video Command) Keyword - VOLUME percentage

**VOLUME percentage**  
Sets the volume level.

---

## SET (Digital Video Command) Keyword - AUDIOSYNC value

**AUDIOSYNC value**  
Sets the audio synchronization value, where *value* is the MMTIME value to adjust audio time ahead relative to video time.

---

## SET (Digital Video Command) Keyword - FORWARD

**FORWARD**

Sets the audio synchronization value, where *value* is the MMTIME value to adjust audio time ahead relative to video time.

-----

## SET (Digital Video Command) Keyword - REVERSE

### **REVERSE**

Sets the audio synchronization value, where *value* is the MMTIME value to adjust audio time back relative to video time.

-----

## SET (Digital Video Command) Keyword - BRIGHTNESS level

### **BRIGHTNESS level**

Sets the capture card brightness to the specified level (0 - 100). The default value is determined by the user through the Setup application; if no value is set by the user in the Setup, a default provided by the particular device-specific physical device driver is used.

-----

## SET (Digital Video Command) Keyword - CHANNELS integer

### **CHANNELS integer**

Sets the number of channels in the audio soundtrack recording (1 = mono, 2 = stereo). The default setting is 1.

-----

## SET (Digital Video Command) Keyword - CONTRAST level

### **CONTRAST level**

Sets the capture card contrast to the specified level (0 - 100). The default value is determined by the user through the Setup application; if no value is set by the user in the Setup, a default provided by the particular device-specific physical device driver is used.

-----

## SET (Digital Video Command) Keyword - HUE level

### **HUE level**

Sets the capture card hue to the specified level (0 - 100). where 0 is maximum green, 100 is maximum red, and 50 is neutral. The default value is determined by the user through the Setup application; if no value is set by the user in the Setup, a default provided by the particular device-specific physical device driver is used.

-----

# SET (Digital Video Command) Keyword - GRAPHIC COLOR integer

## GRAPHIC COLOR integer

Sets the transparent color used as the "chroma-key" on video overlay hardware (has the same effect as specifying the *TRANSPARENT COLOR integer*). The color is set as a numeric value in the range 0 - (*n*-1) where *n* represents the number of colors available. This value only pertains to video overlay devices, such as Video Blaster.

The default value is determined by the user through the Setup page for the digital video device; if a default value is not set by the user, 0x2D becomes the default.

# SET (Digital Video Command) Keyword - MONITOR state

## MONITOR state

Sets monitoring as specified by **state**. When monitoring is turned on, a monitor window is created. The monitor window is similar to that of the playback window. Valid **state** values are:

<b>on</b>	Enables monitoring.
<b>off</b>	Disables monitoring.

# SET (Digital Video Command) Keyword - RECORD AUDIO ON

## RECORD AUDIO ON

Enables audio soundtrack recording. This is the default.

# SET (Digital Video Command) Keyword - RECORD AUDIO OFF

## RECORD AUDIO OFF

Disables audio soundtrack recording.

# SET (Digital Video Command) Keyword - REFERENCE FRAME INTERVAL n



#### **REFERENCE FRAME INTERVAL *n***

Sets the reference frame interval where *n* refers to a reference frame (I-frame) being inserted every *n*th frame. A value of 0 results in no I-frames, 1 causes every frame to be an I-frame, 2 causes every other frame to be an I-frame, and so on. Although there is no upper bound on the reference frame interval, a reference frame interval that does not exceed two seconds produces the best results. The default reference frame interval is every 15th frame (once a second at the default frame rate of 15 frames per second).

---

## **SET (Digital Video Command) Keyword - SATURATION level**

#### **SATURATION level**

Sets the capture card saturation to the specified level (0 - 100). The default value is determined by the user through the Setup application; if no value is set by the user in the Setup, a default provided by the particular device-specific physical device driver is used.

---

## **SET (Digital Video Command) Keyword - SAMPLESPERSEC integer**

#### **SAMPLESPERSEC integer**

Sets the number of waveform samples per second in the audio soundtrack recording. This value is usually 11025, 22050, or 44100. The default is 11025.

---

## **SET (Digital Video Command) Keyword - SPEED FORMAT PERCENTAGE**

#### **SPEED FORMAT PERCENTAGE**

Sets the speed format to percentage (playback only).

---

## **SET (Digital Video Command) Keyword - SPEED FORMAT FPS**

#### **SPEED FORMAT FPS**

Sets the speed format to frames per second (playback only).

---

## **SET (Digital Video Command) Keyword - TIME FORMAT MILLISECONDS**

#### TIME FORMAT MILLISECONDS

Sets the time format to milliseconds. All position information is this format following this command. You can abbreviate milliseconds as **ms**.

-----

## SET (Digital Video Command) Keyword - TIME FORMAT MMTIME

#### TIME FORMAT MMTIME

Sets the time format to MMTIME.

-----

## SET (Digital Video Command) Keyword - TIME FORMAT FRAMES

#### TIME FORMAT FRAMES

Sets time format to frames. All position information is specified in frames following this command. When the device is opened, **frames** is the default mode.

-----

## SET (Digital Video Command) Keyword - TIME FORMAT HMS

#### TIME FORMAT HMS

Sets time format to Hours, Minutes, Seconds. All position information is this format following this command.

-----

## SET (Digital Video Command) Keyword - TIME FORMAT HMSF

#### TIME FORMAT HMSF

Sets time format to Hours, Minutes, Seconds, and Frames. All position information is this format following this command.

-----

## SET (Digital Video Command) Keyword - TRANSPARENT

# COLOR integer

## TRANSPARENT COLOR integer

Sets the transparent color used as the "chroma-key" on video overlay hardware (has the same effect as specifying *GRAPHIC COLOR integer*). The color is set as a numeric value in the range 0 - (*n*-1) where *n* represents the number of colors available. This value only pertains to video overlay devices, such as Video Blaster.

The default value is determined by the user through the Setup page for the digital video device; if a default value is not set by the user, 0x2D becomes the default.

---

# SET (Digital Video Command) Keyword - VIDEO COMPRESSION fourcc

## VIDEO COMPRESSION fourcc

Specifies the FOURCC compression type used for recording motion video. Only symmetric compressors will be enabled for real-time recording. Possible types include:

DIB	Raw (uncompressed format)
ULTI	Ultimotion
RT21	Indeo 2.1
IV31	Indeo 3.1

**Note:** Compressors are not available for FLIC, MPEG, and Indeo 3.2 in this version of OS/2.

---

# SET (Digital Video Command) Keyword - VIDEO RECORD RATE frames per second

## VIDEO RECORD RATE frames per second

Sets the frame rate for recording as an integral number of frames per second. This sets the target record rate, but there is no guarantee this rate will be attained. Drop frame records will be inserted into the output data stream to indicate frames dropped during the record process. The default record frame is rate 15 frames per second.

---

# SET (Digital Video Command) Keyword - VIDEO RECORD FRAME DURATION integer

## VIDEO RECORD FRAME DURATION integer

Sets the frame rate for recording as the time duration of each frame in microseconds. This is useful for setting non-integer frame rates; for example, 12.5 frames per second of a PAL videodisc:  $1000000/12.5 = 80000$  microseconds. The default frame duration is 66,667 microseconds, equivalent to 15 frames per second. The maximum frame duration is 1,000,000 microseconds (1 frame per second), and the minimum frame duration is 33,333 microseconds (30 frames per second).

## SET (Digital Video Command) Keyword - VIDEO QUALITY level

### VIDEO QUALITY level

Sets the compression quality level setting to be sent to the CODEC. This value is in the range 0 - 10,000. Not all CODECs support quality level settings. The default setting for video quality is 5000.

---

## SET (Digital Video Command) Keyword - VIDEO COLOR integer

### VIDEO COLOR integer

Sets transparency color for transparency in video on dual-plane video adapters such as RealMagic. Graphics will be seen wherever the transparency color appears in the video. The color is set as a numeric value in the range  $0 \dots (n - 1)$ . Where  $n$  represents the number of available colors.

---

## SET (Digital Video Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## SET (Digital Video Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## SET (Digital Video Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## AUDIO

Specifies the audio attributes of the device context determined by the ALL, LEFT, RIGHT, OVER, and VOLUME keywords.

### ALL

Applies to both or all of the channels (default).

Specify ON or OFF with the ALL keyword.

#### ON

Enables audio.

#### OFF

Disables audio.

### LEFT

Applies to the left channel.

Specify ON or OFF with the LEFT keyword.

#### ON

Enables audio for the left channel.

#### OFF

Disables audio for the left channel.

### RIGHT

Applies to the right channel.

Specify ON or OFF with the RIGHT keyword.

#### ON

Enables audio for the right channel.

#### OFF

Disables audio for the right channel.

### OVER milliseconds

Applies the change over the specified time period (fade).

### VOLUME percentage

Sets the volume level.

## AUDIOSYNC value

Sets the audio synchronization value, where *value* is the MMTIME value to adjust audio time ahead relative to video time.

### FORWARD

Sets the audio synchronization value, where *value* is the MMTIME value to adjust audio time ahead relative to video time.

### REVERSE

Sets the audio synchronization value, where *value* is the MMTIME value to adjust audio time back relative to video time.

## BRIGHTNESS level

Sets the capture card brightness to the specified level (0 - 100). The default value is determined by the user through the Setup application; if no value is set by the user in the Setup, a default provided by the particular device-specific physical device driver is used.

## CHANNELS integer

Sets the number of channels in the audio soundtrack recording (1 = mono, 2 = stereo). The default setting is 1.

## CONTRAST level

Sets the capture card contrast to the specified level (0 - 100). The default value is determined by the user through the Setup application; if no value is set by the user in the Setup, a default provided by the particular device-specific physical device driver is used.

## HUE level

Sets the capture card hue to the specified level (0 - 100). where 0 is maximum green, 100 is maximum red, and 50 is neutral. The default value is determined by the user through the Setup application; if no value is set by the user in the Setup, a default provided by the particular device-specific physical device driver is used.

## GRAPHIC COLOR integer

Sets the transparent color used as the "chroma-key" on video overlay hardware (has the same effect as specifying the

*TRANSPARENT COLOR integer*). The color is set as a numeric value in the range 0 - (*n*-1) where *n* represents the number of colors available. This value only pertains to video overlay devices, such as Video Blaster.

The default value is determined by the user through the Setup page for the digital video device; if a default value is not set by the user, 0x2D becomes the default.

#### **MONITOR state**

Sets monitoring as specified by **state**. When monitoring is turned on, a monitor window is created. The monitor window is similar to that of the playback window. Valid **state** values are:

<b>on</b>	Enables monitoring.
<b>off</b>	Disables monitoring.

#### **RECORD AUDIO ON**

Enables audio soundtrack recording. This is the default.

#### **RECORD AUDIO OFF**

Disables audio soundtrack recording.

#### **REFERENCE FRAME INTERVAL *n***

Sets the reference frame interval where **n** refers to a reference frame (I-frame) being inserted every *n*th frame. A value of 0 results in no I-frames, 1 causes every frame to be an I-frame, 2 causes every other frame to be an I-frame, and so on. Although there is no upper bound on the reference frame interval, a reference frame interval that does not exceed two seconds produces the best results. The default reference frame interval is every 15th frame (once a second at the default frame rate of 15 frames per second).

#### **SATURATION level**

Sets the capture card saturation to the specified level (0 - 100). The default value is determined by the user through the Setup application; if no value is set by the user in the Setup, a default provided by the particular device-specific physical device driver is used.

#### **SAMPLESPERSEC integer**

Sets the number of waveform samples per second in the audio soundtrack recording. This value is usually 11025, 22050, or 44100. The default is 11025.

#### **SPEED FORMAT PERCENTAGE**

Sets the speed format to percentage (playback only).

#### **SPEED FORMAT FPS**

Sets the speed format to frames per second (playback only).

#### **TIME FORMAT MILLISECONDS**

Sets the time format to milliseconds. All position information is this format following this command. You can abbreviate milliseconds as **ms**.

#### **TIME FORMAT MMTIME**

Sets the time format to MMTIME.

#### **TIME FORMAT FRAMES**

Sets time format to frames. All position information is specified in frames following this command. When the device is opened, **frames** is the default mode.

#### **TIME FORMAT HMS**

Sets time format to Hours, Minutes, Seconds. All position information is this format following this command.

#### **TIME FORMAT HMSF**

Sets time format to Hours, Minutes, Seconds, and Frames. All position information is this format following this command.

#### **TRANSPARENT COLOR integer**

Sets the transparent color used as the "chroma-key" on video overlay hardware (has the same effect as specifying *GRAPHIC COLOR integer*). The color is set as a numeric value in the range 0 - (*n*-1) where *n* represents the number of colors available. This value only pertains to video overlay devices, such as Video Blaster.

The default value is determined by the user through the Setup page for the digital video device; if a default value is not set by the user, 0x2D becomes the default.

#### **VIDEO COMPRESSION fourcc**

Specifies the FOURCC compression type used for recording motion video. Only symmetric compressors will be enabled for real-time recording. Possible types include:

<b>DIB</b>	Raw (uncompressed format)
<b>ULTI</b>	Ultimotion
<b>RT21</b>	Indeo 2.1
<b>IV31</b>	Indeo 3.1

**Note:** Compressors are not available for FLIC, MPEG, and Indeo 3.2 in this version of OS/2.

**VIDEO RECORD RATE frames per second**

Sets the frame rate for recording as an integral number of frames per second. This sets the target record rate, but there is no guarantee this rate will be attained. Drop frame records will be inserted into the output data stream to indicate frames dropped during the record process. The default record frame is rate 15 frames per second.

**VIDEO RECORD FRAME DURATION integer**

Sets the frame rate for recording as the time duration of each frame in microseconds. This is useful for setting non-integer frame rates; for example, 12.5 frames per second of a PAL videodisc:  $1000000/12.5 = 80000$  microseconds. The default frame duration is 66,667 microseconds, equivalent to 15 frames per second. The maximum frame duration is 1,000,000 microseconds (1 frame per second), and the minimum frame duration is 33,333 microseconds (30 frames per second).

**VIDEO QUALITY level**

Sets the compression quality level setting to be sent to the CODEC. This value is in the range 0 - 10,000. Not all CODECs support quality level settings. The default setting for video quality is 5000.

**VIDEO COLOR integer**

Sets transparency color for transparency in video on dual-plane video adapters such as RealMagic. Graphics will be seen wherever the transparency color appears in the video. The color is set as a numeric value in the range 0...(n - 1). Where n represents the number of available colors.

**WAIT**

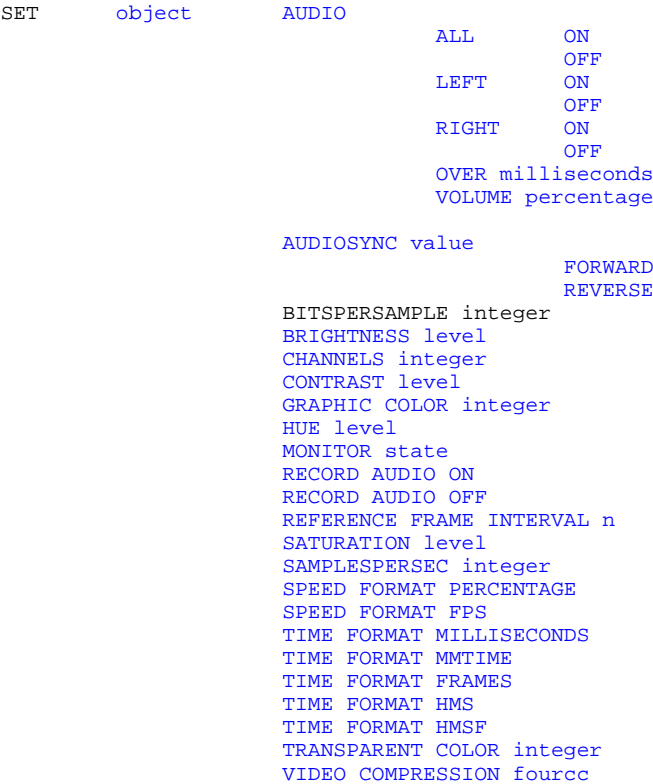
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# SET (Digital Video Command) - Syntax Diagram



```
VIDEO RECORD RATE frames per second
VIDEO RECORD FRAME DURATION integer
VIDEO QUALITY level
VIDEO COLOR integer
```

```
WAIT
NOTIFY
```

## Examples

---

# SET (Digital Video Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## SETTUNER

---

# SETTUNER (Digital Video Command) - Example

```
settuner digitalvideo03 region usacatv tv channel 29
```

---

# SETTUNER (Digital Video Command) - Purpose

The SETTUNER command causes the MCD to change the frequency that the tuner device is tuned to.

---

# SETTUNER (Digital Video Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:



- Device type
- Device name
- Filename
- Alias

-----

## SETTUNER (Digital Video Command) Keyword - TV CHANNEL integer

### TV CHANNEL integer

Sets the channel to the value specified by **integer**. Channel is used along with region and fine-tuning values to calculate the frequency.

-----

## SETTUNER (Digital Video Command) Keyword - REGION name

### REGION name

Sets the region to the value specified by **name**. Region is used along with channel and fine-tuning values to calculate the frequency.

-----

## SETTUNER (Digital Video Command) Keyword - FINETUNE

### FINETUNE

Use the PLUS or MINUS keyword with FINETUNE to indicate whether the fine-tuning value is positive or negative. Fine-tuning is used along with region and channel values to calculate the frequency.

-----

## SETTUNER (Digital Video Command) Keyword - PLUS integer

### PLUS integer

Indicates a *positive* fine-tuning value.

-----

## SETTUNER (Digital Video Command) Keyword - MINUS integer

**MINUS integer**

Indicates a *negative* fine-tuning value.

---

## SETTUNER (Digital Video Command) Keyword - FREQUENCY integer

**FREQUENCY integer**

Sets the frequency being sent to the device driver to the value specified by **integer**. Using the FREQUENCY keyword to directly set the frequency overrides channel, region, and fine-tuning values.

---

## SETTUNER (Digital Video Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## SETTUNER (Digital Video Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## SETTUNER (Digital Video Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**TV CHANNEL integer**

Sets the channel to the value specified by **integer**. Channel is used along with region and fine-tuning values to calculate the frequency.

**REGION name**

Sets the region to the value specified by **name**. Region is used along with channel and fine-tuning values to calculate the frequency.

**FINETUNE**

Use the PLUS or MINUS keyword with FINETUNE to indicate whether the fine-tuning value is positive or negative. Fine-tuning is used along with region and channel values to calculate the frequency.

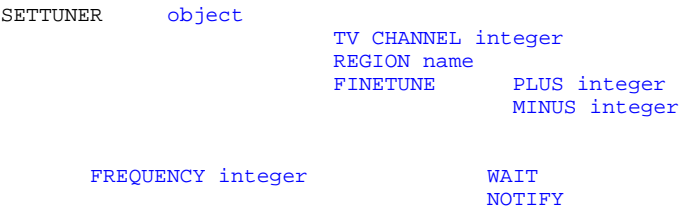
- PLUS integer** Indicates a *positive* fine-tuning value.
- MINUS integer** Indicates a *negative* fine-tuning value.

**FREQUENCY integer**  
Sets the frequency being sent to the device driver to the value specified by **integer**. Using the FREQUENCY keyword to directly set the frequency overrides channel, region, and fine-tuning values.

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

## SETTUNER (Digital Video Command) - Syntax Diagram



Examples

## SETTUNER (Digital Video Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

## STATUS

---

## STATUS (Digital Video Command) - Example

```
status digitalvideo clipboard wait
```

---

## STATUS (Digital Video Command) - Purpose

The STATUS command obtains status information for the device.

---

## STATUS (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## STATUS (Digital Video Command) Keyword - AUDIOSYNC

### **AUDIOSYNC**

Returns the audio synchronization adjust value. This value is always expressed in MMTIME units. The default value is 0.

---

## STATUS (Digital Video Command) Keyword - AUDIOSYNC DIRECTION

### **AUDIOSYNC DIRECTION**

Returns the direction of the adjustment in audio synchronization. This is forward or backward relative to video time. The default is forward.

---

## STATUS (Digital Video Command) Keyword -

# BITSPERSAMPLE

## **BITSPERSAMPLE**

Returns the currently set bits per sample used for playing, recording, and saving.

-----

# STATUS (Digital Video Command) Keyword - BRIGHTNESS

## **BRIGHTNESS**

Returns the brightness level.

-----

# STATUS (Digital Video Command) Keyword - CHANNELS

## **CHANNELS**

Returns the currently set channel count used for playing, recording, and saving.

-----

# STATUS (Digital Video Command) Keyword - CLIPBOARD

## **CLIPBOARD**

Returns TRUE if compatible data is in the clipboard; otherwise, returns FALSE.

-----

# STATUS (Digital Video Command) Keyword - CONTRAST

## **CONTRAST**

Returns the contrast level.

-----

# STATUS (Digital Video Command) Keyword - CURRENT TRACK

## **CURRENT TRACK**

Returns the current track.

---

## STATUS (Digital Video Command) Keyword - DROPPEDFRAMEPCT

### DROPPEDFRAMEPCT

Returns the percentage of dropped frames for playback or recording operations. The value returned is in the range 0-100, where a value of 0 indicates that no frame drops are occurring or have occurred. A value of 100 indicates that all frames are being dropped or have been dropped.

This STATUS value can be queried during a recording operation to obtain the cumulative drops that have occurred since recording began. This value can also be queried during a playback operation to obtain the cumulative frame drops that have occurred since playback began or was resumed after a seek, pause, or stop. If the value is queried when the device is stopped, the percentage of dropped frames accumulated at the end of the last playback or recording operation that was performed is returned.

A value of 0 is returned if no playback or recording operations have been performed, the device is seeking or has been seeked, or the device is playing in scan mode.

---

## STATUS (Digital Video Command) Keyword - GRAPHIC COLOR

### GRAPHIC COLOR

Returns the value of the transparent color used as the "chroma-key" on video overlay hardware.

---

## STATUS (Digital Video Command) Keyword - HORIZONTAL IMAGE EXTENT

### HORIZONTAL IMAGE EXTENT

Not supported.

---

## STATUS (Digital Video Command) Keyword - HORIZONTAL VIDEO EXTENT

### HORIZONTAL VIDEO EXTENT

Returns the horizontal (X) extent of the currently loaded motion video.

---

## STATUS (Digital Video Command) Keyword - HUE

## **HUE**

Returns the hue level.

---

# STATUS (Digital Video Command) Keyword - FORMAT TAG

## **FORMAT TAG**

Returns WAVE\_FORMAT\_PCM, the only format currently supported by the digital video device. However, if a movie is loaded that contains a format other than PCM, the format used in the movie will be returned.

---

# STATUS (Digital Video Command) Keyword - FORWARD

## **FORWARD**

Returns TRUE if the play direction is forward or if the device is not playing.

---

# STATUS (Digital Video Command) Keyword - IMAGE BITSPERPEL

## **IMAGE BITSPERPEL**

Returns the number of bits per pel for saving bit maps.

---

# STATUS (Digital Video Command) Keyword - IMAGE PELFORMAT

## **IMAGE PELFORMAT**

Returns the pel format for saving bit maps or images.

---

# STATUS (Digital Video Command) Keyword - LENGTH

## **LENGTH**

Returns the length in the current time format.

---

## STATUS (Digital Video Command) Keyword - LENGTH TRACK track\_number

### LENGTH TRACK track\_number

Returns the total number of frames in the track specified by **track\_number**.

---

## STATUS (Digital Video Command) Keyword - MEDIA PRESENT

### MEDIA PRESENT

Returns TRUE if the media is inserted in the device; otherwise, the return is FALSE.

---

## STATUS (Digital Video Command) Keyword - MODE

### MODE

Returns **not ready**, **pause**, **play**, **record**, **seek**, or **stop** for the current mode.

---

## STATUS (Digital Video Command) Keyword - MONITOR

### MONITOR

Returns ON or OFF.

---

## STATUS (Digital Video Command) Keyword - MONITOR WINDOW HANDLE

### MONITOR WINDOW HANDLE

Returns the handle of the window used for the monitor window.

---

## STATUS (Digital Video Command) Keyword - NORMAL



# RATE

## **NORMAL RATE**

Returns the normal rate of the currently loaded motion video device element, in the current speed format, either as a percentage or in frames per second. Otherwise, returns 0.

-----

# STATUS (Digital Video Command) Keyword - NUMBER OF TRACKS

## **NUMBER OF TRACKS**

Returns the number of tracks on the media.

-----

# STATUS (Digital Video Command) Keyword - PASTE

## **PASTE**

Returns TRUE if compatible data is to be placed in clipboard; otherwise, returns FALSE.

-----

# STATUS (Digital Video Command) Keyword - POSITION

## **POSITION**

Returns the current position.

-----

# STATUS (Digital Video Command) Keyword - READY

## **READY**

Returns TRUE if the digital video device is ready.

-----

# STATUS (Digital Video Command) Keyword - RECORD AUDIO

## **RECORD AUDIO**

Returns ON or OFF.

---

## STATUS (Digital Video Command) Keyword - REDO

### REDO

Returns TRUE if the redo operation can be performed; otherwise, returns FALSE.

---

## STATUS (Digital Video Command) Keyword - REFERENCE FRAME INTERVAL

### REFERENCE FRAME INTERVAL

Returns the value of  $n$  where  $n$  refers to a reference frame being inserted every  $n$ th frame.

---

## STATUS (Digital Video Command) Keyword - SAMPLESPERSEC

### SAMPLESPERSEC

Returns the currently set samples per second used for playing, recording, and saving.

---

## STATUS (Digital Video Command) Keyword - SATURATION

### SATURATION

Returns the saturation level.

---

## STATUS (Digital Video Command) Keyword - SPEED

### SPEED

Returns the current speed of the device in the currently specified speed format.

---

## STATUS (Digital Video Command) Keyword - SPEED

# FORMAT

## **SPEED FORMAT**

Returns the current speed format of the device.

-----

## STATUS (Digital Video Command) Keyword - TIME FORMAT

### **TIME FORMAT**

Returns the time format.

-----

## STATUS (Digital Video Command) Keyword - TRANSPARENT COLOR

### **TRANSPARENT COLOR**

Returns the value of the transparent color used as the "chroma-key" on video overlay hardware.

-----

## STATUS (Digital Video Command) Keyword - TUNER TV CHANNEL

### **TUNER TV CHANNEL**

Returns the channel that the tuner device is tuned to.

-----

## STATUS (Digital Video Command) Keyword - TUNER HIGH TV CHANNEL

### **TUNER HIGH TV CHANNEL**

Returns the highest channel for the region.

-----

## STATUS (Digital Video Command) Keyword - TUNER LOW TV CHANNEL

#### **TUNER LOW TV CHANNEL**

Returns the lowest channel for the region.

---

## **STATUS (Digital Video Command) Keyword - TUNER FINETUNE**

#### **TUNER FINETUNE**

Returns the fine-tuning value that the tuner device is tuned to.

---

## **STATUS (Digital Video Command) Keyword - TUNER FREQUENCY**

#### **TUNER FREQUENCY**

Returns the frequency value that the tuner device is tuned to.

---

## **STATUS (Digital Video Command) Keyword - UNDO**

#### **UNDO**

Returns TRUE if the undo operation can be performed; otherwise, returns FALSE.

---

## **STATUS (Digital Video Command) Keyword - VALID SIGNAL**

#### **VALID SIGNAL**

Returns TRUE if there is a signal present.

---

## **STATUS (Digital Video Command) Keyword - VERTICAL IMAGE EXTENT**

#### **VERTICAL IMAGE EXTENT**

Not supported.

---

## STATUS (Digital Video Command) Keyword - VERTICAL VIDEO EXTENT

### VERTICAL VIDEO EXTENT

Returns the vertical (Y) extent of the currently loaded motion video.

---

## STATUS (Digital Video Command) Keyword - VIDEO COMPRESSION

### VIDEO COMPRESSION

Returns the current FOURCC compression type used for recording of motion video.

---

## STATUS (Digital Video Command) Keyword - VIDEO RECORD FRAME DURATION

### VIDEO RECORD FRAME DURATION

Returns the frame rate for recording as the time duration of each frame in microseconds.

---

## STATUS (Digital Video Command) Keyword - VIDEO RECORD RATE

### VIDEO RECORD RATE

Returns current rate for recording as an integral number of frames per second.

---

## STATUS (Digital Video Command) Keyword - VIDEO QUALITY

### VIDEO QUALITY

Returns the motion video quality level.

-----

## STATUS (Digital Video Command) Keyword - VIDEO COLOR

### VIDEO COLOR

Returns the value of the transparent color used as the "chroma-key" on video overlay hardware.

-----

## STATUS (Digital Video Command) Keyword - VOLUME

### VOLUME

Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

-----

## STATUS (Digital Video Command) Keyword - WINDOW HANDLE

### WINDOW HANDLE

Returns the handle of the window used for the digital video return value.

-----

## STATUS (Digital Video Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

-----

## STATUS (Digital Video Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## STATUS (Digital Video Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**AUDIOSYNC**

Returns the audio synchronization adjust value. This value is always expressed in MMTIME units. The default value is 0.

**AUDIOSYNC DIRECTION**

Returns the direction of the adjustment in audio synchronization. This is forward or backward relative to video time. The default is forward.

**BITSPERSAMPLE**

Returns the currently set bits per sample used for playing, recording, and saving.

**BRIGHTNESS**

Returns the brightness level.

**CHANNELS**

Returns the currently set channel count used for playing, recording, and saving.

**CLIPBOARD**

Returns TRUE if compatible data is in the clipboard; otherwise, returns FALSE.

**CONTRAST**

Returns the contrast level.

**CURRENT TRACK**

Returns the current track.

**DROPPEDFRAMEPCT**

Returns the percentage of dropped frames for playback or recording operations. The value returned is in the range 0-100, where a value of 0 indicates that no frame drops are occurring or have occurred. A value of 100 indicates that all frames are being dropped or have been dropped.

This STATUS value can be queried during a recording operation to obtain the cumulative drops that have occurred since recording began. This value can also be queried during a playback operation to obtain the cumulative frame drops that have occurred since playback began or was resumed after a seek, pause, or stop. If the value is queried when the device is stopped, the percentage of dropped frames accumulated at the end of the last playback or recording operation that was performed is returned.

A value of 0 is returned if no playback or recording operations have been performed, the device is seeking or has been seeked, or the device is playing in scan mode.

**GRAPHIC COLOR**

Returns the value of the transparent color used as the "chroma-key" on video overlay hardware.

**HORIZONTAL IMAGE EXTENT**

Not supported.

**HORIZONTAL VIDEO EXTENT**

Returns the horizontal (X) extent of the currently loaded motion video.

**HUE**

Returns the hue level.

**FORMAT TAG**

Returns WAVE\_FORMAT\_PCM, the only format currently supported by the digital video device. However, if a movie is loaded that contains a format other than PCM, the format used in the movie will be returned.

**FORWARD**

Returns TRUE if the play direction is forward or if the device is not playing.

**IMAGE BITSPERPEL**

Returns the number of bits per pel for saving bit maps.

**IMAGE PELFORMAT**

Returns the pel format for saving bit maps or images.

**LENGTH**

Returns the length in the current time format.

**LENGTH TRACK *track\_number***

Returns the total number of frames in the track specified by ***track\_number***.

**MEDIA PRESENT**

Returns TRUE if the media is inserted in the device; otherwise, the return is FALSE.

**MODE**

Returns **not ready**, **pause**, **play**, **record**, **seek**, or **stop** for the current mode.

**MONITOR**

Returns ON or OFF.

**MONITOR WINDOW HANDLE**

Returns the handle of the window used for the monitor window.

**NORMAL RATE**

Returns the normal rate of the currently loaded motion video device element, in the current speed format, either as a percentage or in frames per second. Otherwise, returns 0.

**NUMBER OF TRACKS**

Returns the number of tracks on the media.

**PASTE**

Returns TRUE if compatible data is to be placed in clipboard; otherwise, returns FALSE.

**POSITION**

Returns the current position.

**READY**

Returns TRUE if the digital video device is ready.

**RECORD AUDIO**

Returns ON or OFF.

**REDO**

Returns TRUE if the redo operation can be performed; otherwise, returns FALSE.

**REFERENCE FRAME INTERVAL**

Returns the value of *n* where *n* refers to a reference frame being inserted every *n*th frame.

**SAMPLESPERSEC**

Returns the currently set samples per second used for playing, recording, and saving.

**SATURATION**

Returns the saturation level.

**SPEED**

Returns the current speed of the device in the currently specified speed format.

**SPEED FORMAT**

Returns the current speed format of the device.

**TIME FORMAT**

Returns the time format.

**TRANSPARENT COLOR**

Returns the value of the transparent color used as the "chroma-key" on video overlay hardware.

**TUNER TV CHANNEL**

Returns the channel that the tuner device is tuned to.

**TUNER HIGH TV CHANNEL**

Returns the highest channel for the region.

**TUNER LOW TV CHANNEL**

Returns the lowest channel for the region.

**TUNER FINETUNE**

Returns the fine-tuning value that the tuner device is tuned to.



- TUNER FREQUENCY**  
Returns the frequency value that the tuner device is tuned to.
- UNDO**  
Returns TRUE if the undo operation can be performed; otherwise, returns FALSE.
- VALID SIGNAL**  
Returns TRUE if there is a signal present.
- VERTICAL IMAGE EXTENT**  
Not supported.
- VERTICAL VIDEO EXTENT**  
Returns the vertical (Y) extent of the currently loaded motion video.
- VIDEO COMPRESSION**  
Returns the current FOURCC compression type used for recording of motion video.
- VIDEO RECORD FRAME DURATION**  
Returns the frame rate for recording as the time duration of each frame in microseconds.
- VIDEO RECORD RATE**  
Returns current rate for recording as an integral number of frames per second.
- VIDEO QUALITY**  
Returns the motion video quality level.
- VIDEO COLOR**  
Returns the value of the transparent color used as the "chroma-key" on video overlay hardware.
- VOLUME**  
Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.
- WINDOW HANDLE**  
Returns the handle of the window used for the digital video return value.
- WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.
- NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## STATUS (Digital Video Command) - Syntax Diagram

STATUS	object	AUDIOSYNC	
		AUDIOSYNC DIRECTION	WAIT
		BITSPERSAMPLE	NOTIFY
		IMAGE BITSPERPEL	
		IMAGE PELFORMAT	
		BRIGHTNESS	
		CHANNELS	
		CLIPBOARD	
		CONTRAST	
		CURRENT TRACK	
		DROPPEDFRAMEPCT	
		GRAPHIC COLOR	
		HORIZONTAL IMAGE EXTENT	
		HORIZONTAL VIDEO EXTENT	
		HUE	
		FORMAT TAG	

FORWARD  
LENGTH  
LENGTH TRACK track\_number  
MEDIA PRESENT  
MODE  
MONITOR  
MONITOR WINDOW HANDLE  
PASTE  
POSITION  
NORMAL RATE  
NUMBER OF TRACKS  
READY  
RECORD AUDIO  
REDO  
REFERENCE FRAME INTERVAL  
SATURATION  
SAMPLESPERSEC  
SPEED  
SPEED FORMAT  
TIME FORMAT  
TRANSPARENT COLOR  
TUNER TV CHANNEL  
TUNER HIGH TV CHANNEL  
TUNER LOW TV CHANNEL  
TUNER FINETUNE  
TUNER FREQUENCY  
UNDO  
VALID SIGNAL  
VERTICAL IMAGE EXTENT  
VERTICAL VIDEO EXTENT  
VIDEO COMPRESSION  
VIDEO RECORD RATE  
VIDEO RECORD FRAME DURATION  
VIDEO QUALITY  
VIDEO COLOR  
VOLUME  
WINDOW HANDLE

Examples

---

## STATUS (Digital Video Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## STEP

---

## STEP (Digital Video Command) - Example

step digitalvideo reverse wait

-----

## STEP (Digital Video Command) - Purpose

The STEP command steps the play one or more frames forward or reverse. The default action is to step one frame forward.

-----

## STEP (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## STEP (Digital Video Command) Keyword - REVERSE

### **REVERSE**

Steps the frames in reverse. Only steps to I-frames.

-----

## STEP (Digital Video Command) Keyword - BY frames

### **BY frames**

Indicates the number of frames to step.

-----

## STEP (Digital Video Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

# STEP (Digital Video Command) Keyword - NOTIFY

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# STEP (Digital Video Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## REVERSE

Steps the frames in reverse. Only steps to I-frames.

## BY frames

Indicates the number of frames to step.

## WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# STEP (Digital Video Command) - Syntax Diagram

STEP      [object](#)      [REVERSE](#)  
                                 [BY frames](#)      [WAIT](#)  
   [NOTIFY](#)



-----

# STEP (Digital Video Command) - Remarks

If you are using an application-defined window and your application is running on a system without direct-access device driver support for motion video, do *not* specify WAIT with the STEP command unless the thread issuing the message is separate from the thread reading the

message queue.

---

## STEP (Digital Video Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Remarks](#)  
[Example](#)  
[Glossary](#)

---

## UNDO

---

## UNDO (Digital Video Command) - Example

```
open macaw.avi alias vid shareable wait
delete vid to 8000 wait
undo digitalvideo wait
```

---

## UNDO (Digital Video Command) - Purpose

The UNDO command undoes the last editing (delete, cut, or paste) change to a file. The position of the media after the undo is 0. Multiple UNDO operations are permitted to restore each previously performed editing operation.

---

## UNDO (Digital Video Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## UNDO (Digital Video Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

# UNDO (Digital Video Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# UNDO (Digital Video Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**WAIT**

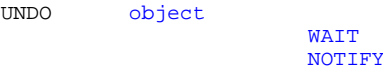
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# UNDO (Digital Video Command) - Syntax Diagram



-----

# UNDO (Digital Video Command) - Topics

Select an item:

[Purpose](#)

[Syntax Diagram](#)

[Keywords](#)

[Example](#)

[Glossary](#)

---

## WHERE

---

### WHERE (Digital Video Command) - Example

```
where digitalvideo window monitor wait
```

---

### WHERE (Digital Video Command) - Purpose

The WHERE command obtains a rectangle array specifying the source or destination area.

---

### WHERE (Digital Video Command) Keyword - object

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

### WHERE (Digital Video Command) Keyword - DESTINATION

#### **DESTINATION**

Requests the position and size of playback video relative to playback window.

---

### WHERE (Digital Video Command) Keyword - MONITOR

# DESTINATION

## MONITOR DESTINATION

Requests the position and size of monitor video relative to monitor window.

-----

# WHERE (Digital Video Command) Keyword - RECORD DESTINATION

## RECORD DESTINATION

Requests the size of movie to be recorded. The coordinates returned are those previously set with the [PUT](#) object **RECORD DESTINATION AT rect** command.

-----

# WHERE (Digital Video Command) Keyword - ADJUSTED

## ADJUSTED

Returns the coordinates that will actually be used to record a movie or get an image buffer based on what was previously set with the [PUT](#) command *and* the capabilities of the capture hardware.

An application can use the WHERE command with and without the ADJUSTED keyword to see the effect of the multiple-integral rule imposed by capture cards that cannot distort. See [PUT](#) for more information on the multiple-integral rule.

-----

# WHERE (Digital Video Command) Keyword - RECORD SOURCE

## RECORD SOURCE

Requests the position and size of source video relative to video source extent. The coordinates returned are those previously set with the [PUT](#) object **RECORD SOURCE AT rect** command.

-----

# WHERE (Digital Video Command) Keyword - ADJUSTED

## ADJUSTED

Returns the coordinates of the source rectangle based on what was previously set with the [PUT](#) command *and* the capabilities of the capture hardware.

An application can use the WHERE command with and without the ADJUSTED keyword to see the effect of the multiple-integral rule imposed by capture cards that cannot distort. See [PUT](#) for more information on the multiple-integral rule.



---

## WHERE (Digital Video Command) Keyword - WINDOW

### **WINDOW**

Requests the position and size of playback window relative to its parent.

---

## WHERE (Digital Video Command) Keyword - MONITOR

### **MONITOR**

Requests the window size and position for the monitor window.

---

## WHERE (Digital Video Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## WHERE (Digital Video Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## WHERE (Digital Video Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### **DESTINATION**

Requests the position and size of playback video relative to playback window.

### **MONITOR DESTINATION**

Requests the position and size of monitor video relative to monitor window.

**RECORD DESTINATION**

Requests the size of movie to be recorded. The coordinates returned are those previously set with the **PUT object RECORD DESTINATION AT rect** command.

**ADJUSTED**

Returns the coordinates that will actually be used to record a movie or get an image buffer based on what was previously set with the **PUT** command *and* the capabilities of the capture hardware.

An application can use the WHERE command with and without the ADJUSTED keyword to see the effect of the multiple-integral rule imposed by capture cards that cannot distort. See **PUT** for more information on the multiple-integral rule.

**RECORD SOURCE**

Requests the position and size of source video relative to video source extent. The coordinates returned are those previously set with the **PUT object RECORD SOURCE AT rect** command.

**ADJUSTED**

Returns the coordinates of the source rectangle based on what was previously set with the **PUT** command *and* the capabilities of the capture hardware.

An application can use the WHERE command with and without the ADJUSTED keyword to see the effect of the multiple-integral rule imposed by capture cards that cannot distort. See **PUT** for more information on the multiple-integral rule.

**WINDOW**

Requests the position and size of playback window relative to its parent.

**MONITOR**

Requests the window size and position for the monitor window.

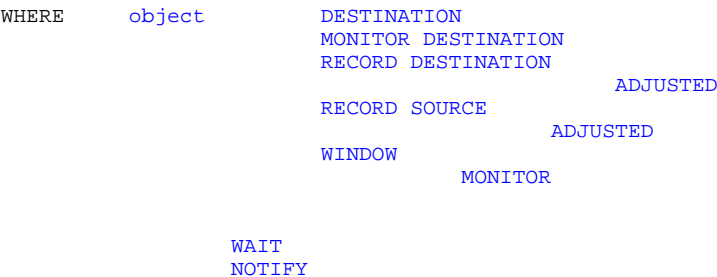
**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

WHERE (Digital Video Command) - Syntax Diagram



Examples

WHERE (Digital Video Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## WINDOW

---

### WINDOW (Digital Video Command) - Example

```
window digitalvideo handle default wait
```

---

### WINDOW (Digital Video Command) - Purpose

The WINDOW command specifies the window and the window characteristics that a graphic device should use for display.

---

### WINDOW (Digital Video Command) Keyword - object

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

### WINDOW (Digital Video Command) Keyword - HANDLE window\_handle

#### **HANDLE window\_handle**

Specifies the handle of the destination window used as an alternate to the default window.

---

## WINDOW (Digital Video Command) Keyword - HANDLE DEFAULT

### **HANDLE DEFAULT**

Specifies the digital video driver should create and manage its own window. This flag can be used to set the display back to the driver's default window.

---

## WINDOW (Digital Video Command) Keyword - STATE HIDE

### **STATE HIDE**

Hides the current display window. Default window or monitor window only.

---

## WINDOW (Digital Video Command) Keyword - STATE MAXIMIZE

### **STATE MAXIMIZE**

Maximizes current display window. Default window or monitor window only.

---

## WINDOW (Digital Video Command) Keyword - STATE MINIMIZE

### **STATE MINIMIZE**

Minimizes the specified window and activates the top-level window in the window manager's list. Default window or monitor window only.

---

## WINDOW (Digital Video Command) Keyword - STATE MINIMIZED

### **STATE MINIMIZED**

Minimizes the current display window. Default window only or monitor window only.

---

# WINDOW (Digital Video Command) Keyword - STATE DEACTIVATE

**STATE DEACTIVATE**  
Displays a window in its current state. The window that is currently active remains active. Default window or monitor window only.

-----

# WINDOW (Digital Video Command) Keyword - STATE ACTIVATE

**STATE ACTIVATE**  
Displays a window in its most recent size and state. The window that is currently active remains active. Default window or monitor window only.

-----

# WINDOW (Digital Video Command) Keyword - STATE RESTORE

**STATE RESTORE**  
Displays the current display window as it was created. Default window or monitor window only.

-----

# WINDOW (Digital Video Command) Keyword - STATE SHOW

**STATE SHOW**  
Shows the current display window. Default window or monitor window only.

-----

# WINDOW (Digital Video Command) Keyword - TEXT caption

**TEXT caption**  
Specifies the **caption** for the display window. Default window only.

-----

# WINDOW (Digital Video Command) Keyword - MONITOR

#### **MONITOR**

Specifies the functions associated with the WINDOW command are to be applied to the monitor window. The monitor window output can be directed to an application-specified window in the same manner as video playback.

**Note:** This keyword must be last in the string command sequence but precede the WAIT or NOTIFY keywords.

-----

## WINDOW (Digital Video Command) Keyword - WAIT

#### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## WINDOW (Digital Video Command) Keyword - NOTIFY

#### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## WINDOW (Digital Video Command) - Keywords

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

#### **HANDLE window\_handle**

Specifies the handle of the destination window used as an alternate to the default window.

#### **HANDLE DEFAULT**

Specifies the digital video driver should create and manage its own window. This flag can be used to set the display back to the driver's default window.

#### **STATE HIDE**

Hides the current display window. Default window or monitor window only.

#### **STATE MAXIMIZE**

Maximizes current display window. Default window or monitor window only.

#### **STATE MINIMIZE**

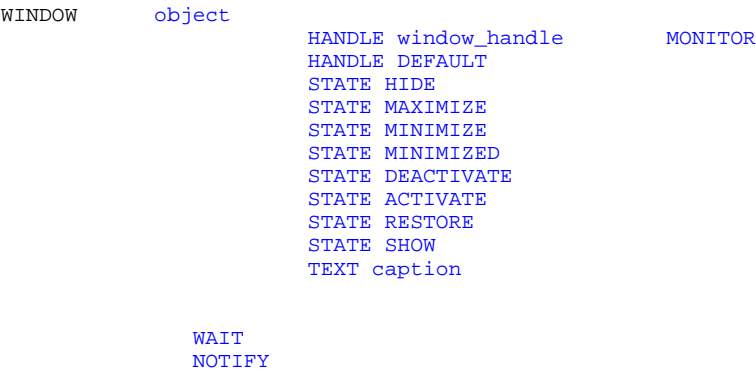
Minimizes the specified window and activates the top-level window in the window manager's list. Default window or monitor window only.

#### **STATE MINIMIZED**

Minimizes the current display window. Default window only or monitor window only.

- STATE DEACTIVATE**  
Displays a window in its current state. The window that is currently active remains active. Default window or monitor window only.
- STATE ACTIVATE**  
Displays a window in its most recent size and state. The window that is currently active remains active. Default window or monitor window only.
- STATE RESTORE**  
Displays the current display window as it was created. Default window or monitor window only.
- STATE SHOW**  
Shows the current display window. Default window or monitor window only.
- TEXT caption**  
Specifies the **caption** for the display window. Default window only.
- MONITOR**  
Specifies the functions associated with the WINDOW command are to be applied to the monitor window. The monitor window output can be directed to an application-specified window in the same manner as video playback.  
  
**Note:** This keyword must be last in the string command sequence but precede the WAIT or NOTIFY keywords.
- WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.
- NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_McINotify message is sent to the application window procedure.

## WINDOW (Digital Video Command) - Syntax Diagram



## WINDOW (Digital Video Command) - Topics

Select an item:  
[Purpose](#)

---

# MIDI Sequencer Commands

The MIDI sequencer device supports the device-type specific command, [CUE](#), and extensions to the following basic and required commands:

- [CAPABILITY](#)
- [CLOSE](#)
- [CONNECTOR](#)
- [CUE](#)
- [SET](#)
- [STATUS](#)

---

## CAPABILITY

---

### CAPABILITY (MIDI Command) - Example

The following command returns FALSE.

```
capability sequencer can record wait
```

---

### CAPABILITY (MIDI Command) - Purpose

The CAPABILITY command requests additional information about the capabilities of the MIDI sequencer.

---

### CAPABILITY (MIDI Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias



---

## CAPABILITY (MIDI Command) Keyword - CAN EJECT

### CAN EJECT

Returns FALSE. The sequencer cannot eject the media.

---

## CAPABILITY (MIDI Command) Keyword - CAN PLAY

### CAN PLAY

Returns TRUE if the sequencer can play the media.

---

## CAPABILITY (MIDI Command) Keyword - CAN RECORD

### CAN RECORD

Returns FALSE. The sequencer cannot record MIDI data.

---

## CAPABILITY (MIDI Command) Keyword - CAN SAVE

### CAN SAVE

Returns TRUE if the sequencer can save MIDI data.

---

## CAPABILITY (MIDI Command) Keyword - CAN SETVOLUME

### CAN SETVOLUME

Returns TRUE if the device supports software control of volume level.

---

## CAPABILITY (MIDI Command) Keyword - COMPOUND DEVICE

### COMPOUND DEVICE

Returns TRUE.

---

## CAPABILITY (MIDI Command) Keyword - DEVICE TYPE

### DEVICE TYPE

Returns **sequencer**.

---

## CAPABILITY (MIDI Command) Keyword - HAS AUDIO

### HAS AUDIO

Returns TRUE. The sequencer supports playback.

---

## CAPABILITY (MIDI Command) Keyword - HAS VIDEO

### HAS VIDEO

Returns FALSE. The sequencer does not support video.

---

## CAPABILITY (MIDI Command) Keyword - MESSAGE command

### MESSAGE command

Returns TRUE if the device supports the command specified by **command**. The **command** can be any string command such as OPEN, PLAY, and so on.

---

## CAPABILITY (MIDI Command) Keyword - PREROLL TIME

### PREROLL TIME

Returns 0, indicating the preroll time is not bounded.

---

## CAPABILITY (MIDI Command) Keyword - PREROLL TYPE

#### **PREROLL TYPE**

Returns the preroll characteristics of the device: Returns NOTIFIED.

---

## **CAPABILITY (MIDI Command) Keyword - USES FILES**

#### **USES FILES**

Returns TRUE. The sequencer uses files for operation.

---

## **CAPABILITY (MIDI Command) Keyword - WAIT**

#### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

---

## **CAPABILITY (MIDI Command) Keyword - NOTIFY**

#### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## **CAPABILITY (MIDI Command) - Keywords**

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

#### **CAN EJECT**

Returns FALSE. The sequencer cannot eject the media.

#### **CAN PLAY**

Returns TRUE if the sequencer can play the media.

#### **CAN RECORD**

Returns FALSE. The sequencer cannot record MIDI data.

#### **CAN SAVE**

Returns TRUE if the sequencer can save MIDI data.

**CAN SETVOLUME**

Returns TRUE if the device supports software control of volume level.

**COMPOUND DEVICE**

Returns TRUE.

**DEVICE TYPE**

Returns **sequencer**.

**HAS AUDIO**

Returns TRUE. The sequencer supports playback.

**HAS VIDEO**

Returns FALSE. The sequencer does not support video.

**MESSAGE command**

Returns TRUE if the device supports the command specified by **command**. The **command** can be any string command such as OPEN, PLAY, and so on.

**PREROLL TIME**

Returns 0, indicating the preroll time is not bounded.

**PREROLL TYPE**

Returns the preroll characteristics of the device: Returns NOTIFIED.

**USES FILES**

Returns TRUE. The sequencer uses files for operation.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

-----

## CAPABILITY (MIDI Command) - Syntax Diagram

CAPABILITY	<b>object</b>	CAN EJECT	
		CAN PLAY	WAIT
		CAN RECORD	NOTIFY
		CAN SAVE	
		CAN SETVOLUME	
		COMPOUND DEVICE	
		DEVICE TYPE	
		HAS AUDIO	
		HAS VIDEO	
		MESSAGE command	
		PREROLL TYPE	
		PREROLL TIME	
		USES FILES	

Examples

-----

## CAPABILITY (MIDI Command) - Topics

Select an item:

[Purpose](#)

[Syntax Diagram](#)

[Keywords](#)

[Example](#)

[Glossary](#)

---

## CLOSE

---

### CLOSE (MIDI Command) - Example

```
close sequencer wait
```

---

### CLOSE (MIDI Command) - Purpose

The CLOSE command closes the sequencer element and the port and file associated with it. When the last element is closed, MCI unloads the sequencer.

---

### CLOSE (MIDI Command) Keyword - object

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

### CLOSE (MIDI Command) Keyword - WAIT

#### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

# CLOSE (MIDI Command) Keyword - NOTIFY

**NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CLOSE (MIDI Command) - Keywords

**object**  
Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CLOSE (MIDI Command) - Syntax Diagram

CLOSE      [object](#)

[WAIT](#)  
[NOTIFY](#)



## CLOSE (MIDI Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

# CONNECTOR

---

## CONNECTOR (MIDI Command) - Example

```
connector sequencer enable type line out
```

---

## CONNECTOR (MIDI Command) - Purpose

The CONNECTOR command enables, disables, or queries the status of connectors on a device.

---

## CONNECTOR (MIDI Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CONNECTOR (MIDI Command) Keyword - ENABLE

### **ENABLE**

Enables information flow through the indicated connector. Using this option requires that the NUMBER or TYPE keywords, or both also be specified.

---

## CONNECTOR (MIDI Command) Keyword - DISABLE

### **DISABLE**

Disables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

-----

## CONNECTOR (MIDI Command) Keyword - QUERY

### QUERY

Queries the status of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Using this option requires that the NUMBER or TYPE keywords, or both also be specified.

-----

## CONNECTOR (MIDI Command) Keyword - NUMBER connector\_number

### NUMBER connector\_number

The connector number on which to perform the requested action. If the TYPE keyword is included, the connector number is interpreted as a relative offset within the specified connector type.

-----

## CONNECTOR (MIDI Command) Keyword - TYPE connector\_type

### TYPE connector\_type

Indicates the type of connector to which the requested action applies. The following connector types are directly supported by this device:

MIDI stream

Digital input or output for the audio amplifier/mixer. This connector is always enabled.

The MIDI sequencer device also recognizes the following connector types and will attempt to control the corresponding amp/mixer connector if the amp/mixer provides the support.

line out

The line output connector. This connector is usually attached to the line in connector of another device such as a tape recorder or other audio device.

speakers

The speakers connector. This connector is usually attached to a pair of external or internal speakers.

headphones

The headphones connector. This connector is usually attached to a pair of headphones.

-----

## CONNECTOR (MIDI Command) Keyword - WAIT



**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## CONNECTOR (MIDI Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CONNECTOR (MIDI Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**ENABLE**

Enables information flow through the indicated connector. Using this option requires that the NUMBER or TYPE keywords, or both also be specified.

**DISABLE**

Disables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

**QUERY**

Queries the status of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Using this option requires that the NUMBER or TYPE keywords, or both also be specified.

**NUMBER connector\_number**

The connector number on which to perform the requested action. If the TYPE keyword is included, the connector number is interpreted as a relative offset within the specified connector type.

**TYPE connector\_type**

Indicates the type of connector to which the requested action applies. The following connector types are directly supported by this device:

MIDI stream

Digital input or output for the audio amplifier/mixer. This connector is always enabled.

The MIDI sequencer device also recognizes the following connector types and will attempt to control the corresponding amp/mixer connector if the amp/mixer provides the support.

line out

The line output connector. This connector is usually attached to the line in connector of another device such as a tape recorder or other audio device.

speakers

The speakers connector. This connector is usually attached to a pair of external or internal speakers.

headphones

The headphones connector. This connector is usually attached to a pair of headphones.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the

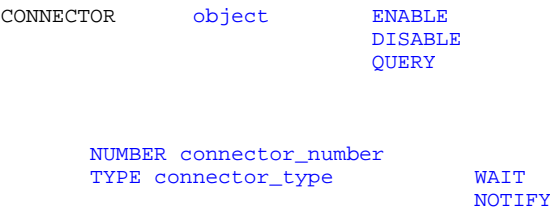
application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# CONNECTOR (MIDI Command) - Syntax Diagram



---

# CONNECTOR (MIDI Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

# CUE

---

# CUE (MIDI Command) - Example

```
open sequencer01 alias seq wait
load seq sounds.mid wait
cue seq output wait
```

# CUE (MIDI Command) - Purpose

The CUE command prepares for playback or recording. The CUE command does not have to be issued prior to playback or recording; however, depending on the device, it might reduce the delay associated with the PLAY or RECORD command. The command fails if playing or recording is in progress.

The CUE command is not related to the [SETCUEPOINT](#) command.

---

## CUE (MIDI Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CUE (MIDI Command) Keyword - INPUT

### **INPUT**

Prepares the device for recording.

---

## CUE (MIDI Command) Keyword - OUTPUT

### **OUTPUT**

Prepares the device for playback.

---

## CUE (MIDI Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## CUE (MIDI Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# CUE (MIDI Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**INPUT**

Prepares the device for recording.

**OUTPUT**

Prepares the device for playback.

**WAIT**

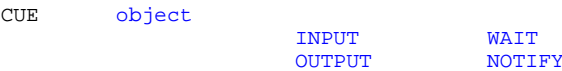
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# CUE (MIDI Command) - Syntax Diagram



---

# CUE (MIDI Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)

---

## SET

---

### SET (MIDI Command) - Example

```
set sequencer time format ms wait
```

---

### SET (MIDI Command) - Purpose

The SET command sets the various control items.

---

### SET (MIDI Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

### SET (MIDI Command) Keyword - AUDIO

**AUDIO**

Sets the audio attributes of the device context specified by the ALL, LEFT, RIGHT, OVER and VOLUME keywords.

---

### SET (MIDI Command) Keyword - ALL

**ALL**

Applies to both or all of the channels (default).

Specify ON or OFF with the ALL keyword.

-----

## SET (MIDI Command) Keyword - ON

**ON**

Enables audio output.

-----

## SET (MIDI Command) Keyword - OFF

**OFF**

Disables audio output.

-----

## SET (MIDI Command) Keyword - LEFT

**LEFT**

Applies to left channel.

Specify ON or OFF with the LEFT keyword.

-----

## SET (MIDI Command) Keyword - ON

**ON**

Enables audio output to the left channel.

-----

## SET (MIDI Command) Keyword - OFF

**OFF**

Disables audio output to the left channel.

-----

## SET (MIDI Command) Keyword - RIGHT

**RIGHT**

Applies to right channel.

Specify ON or OFF with the RIGHT keyword.

---

## SET (MIDI Command) Keyword - ON

**ON**

Enables audio output to the right channel.

---

## SET (MIDI Command) Keyword - OFF

**OFF**

Disables audio output to the right channel.

---

## SET (MIDI Command) Keyword - OVER milliseconds

**OVER milliseconds**

Applies the change over the specified time period (fade).

---

## SET (MIDI Command) Keyword - VOLUME percentage

**VOLUME percentage**

Sets the device/mixer channel volume level.

---

## SET (MIDI Command) Keyword - MASTER MIDI

**MASTER MIDI**

Sets the MIDI sequencer as the synchronization source. Synchronization data is sent in MIDI format. The IBM sequencer does not support this option.

---

## SET (MIDI Command) Keyword - MASTER NONE

### **MASTER NONE**

Inhibits the sequencer from sending synchronization data. The IBM sequencer does not support this option.

---

## SET (MIDI Command) Keyword - MASTER SMPTE

### **MASTER SMPTE**

Sets the MIDI sequencer as the synchronization source. Synchronization data is sent in SMPTE format. The IBM sequencer does not support this option.

---

## SET (MIDI Command) Keyword - OFFSET time

### **OFFSET time**

The SMPTE offset *time* in colon form (HOURS:MINUTES:SECONDS:FRAMES).

---

## SET (MIDI Command) Keyword - PORT MAPPER

### **PORT MAPPER**

The MIDI mapper is the port receiving the MIDI messages. This command will fail if the MIDI mapper or a port it needs is being used by another sequence or client.

---

## SET (MIDI Command) Keyword - PORT NONE

### **PORT NONE**

Disables the sending of MIDI messages. Currently this function is not supported by the IBM sequencer.

---

## SET (MIDI Command) Keyword - SLAVE FILE

### **SLAVE FILE**

Sets the MIDI sequencer to use file data as the synchronization source. This is the default.



---

## SET (MIDI Command) Keyword - SLAVE MIDI

### SLAVE MIDI

Sets the MIDI sequencer to use incoming MIDI data as the synchronization source. The sequencer recognizes synchronization data with the MIDI format. The IBM sequencer does not support this option.

---

## SET (MIDI Command) Keyword - SLAVE NONE

### SLAVE NONE

Sets the MIDI sequencer to ignore synchronization data.

---

## SET (MIDI Command) Keyword - SLAVE SMPTE

### SLAVE SMPTE

Sets the MIDI sequencer to use incoming MIDI data for the synchronization source. The sequencer recognizes synchronization data with the SMPTE format. The IBM sequencer does not support this option.

---

## SET (MIDI Command) Keyword - TEMPO integer

### TEMPO integer

The tempo of the sequence according to the current time format. For a ppqn-based file, the **integer** is interpreted as beats per minute. For a SMPTE based file, the **integer** is interpreted as frames per second.

---

## SET (MIDI Command) Keyword - TIME FORMAT MILLISECONDS

### TIME FORMAT MILLISECONDS

Sets time format to milliseconds. All position information is specified as milliseconds following this command. The sequence file sets the default format to ppqn or SMPTE. You can abbreviate milliseconds as **ms**.

---

## SET (MIDI Command) Keyword - TIME FORMAT MMTIME

**TIME FORMAT MMTIME**

Sets the time format to MMTIME.

---

## SET (MIDI Command) Keyword - TIME FORMAT SONG POINTER

**TIME FORMAT SONG POINTER**

Sets the time format to song-pointer (sixteenth notes). This can only be performed for a sequence of division type ppqn. Currently this function is not supported by the IBM sequencer.

---

## SET (MIDI Command) Keyword - TIME FORMAT SMPTE 24

**TIME FORMAT SMPTE 24**

Sets the time format to SMPTE 24 frame rate. All position information is specified in SMPTE format following this command. The sequence file sets the default format to ppqn or SMPTE.

---

## SET (MIDI Command) Keyword - TIME FORMAT SMPTE 25

**TIME FORMAT SMPTE 25**

Sets the time format to SMPTE 25 frame rate. All position information is specified in SMPTE format following this command. The sequence file sets the default format to ppqn or SMPTE.

---

## SET (MIDI Command) Keyword - TIME FORMAT SMPTE 30

**TIME FORMAT SMPTE 30**

Sets the time format to SMPTE 30 frame rate. All position information is specified in SMPTE format following this command. The sequence file sets the default format to ppqn or SMPTE.

---

## SET (MIDI Command) Keyword - TIME FORMAT SMPTE 30 DROP

#### TIME FORMAT SMPTE 30 DROP

Sets the time format to SMPTE 30 drop frame rate. All position information is specified in SMPTE format following this command. The sequence file sets the default format to ppqn or SMPTE. Currently this function is not supported by the IBM sequencer.

-----

## SET (MIDI Command) Keyword - WAIT

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## SET (MIDI Command) Keyword - NOTIFY

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## SET (MIDI Command) - Keywords

#### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

#### AUDIO

Sets the audio attributes of the device context specified by the ALL, LEFT, RIGHT, OVER and VOLUME keywords.

##### ALL

Applies to both or all of the channels (default).

Specify ON or OFF with the ALL keyword.

##### ON

Enables audio output.

##### OFF

Disables audio output.

##### LEFT

Applies to left channel.

Specify ON or OFF with the LEFT keyword.

##### ON

Enables audio output to the left channel.

##### OFF

Disables audio output to the left channel.

## **RIGHT**

Applies to right channel.

Specify ON or OFF with the RIGHT keyword.

### **ON**

Enables audio output to the right channel.

### **OFF**

Disables audio output to the right channel.

## **OVER milliseconds**

Applies the change over the specified time period (fade).

## **VOLUME percentage**

Sets the device/mixer channel volume level.

## **MASTER MIDI**

Sets the MIDI sequencer as the synchronization source. Synchronization data is sent in MIDI format. The IBM sequencer does not support this option.

## **MASTER NONE**

Inhibits the sequencer from sending synchronization data. The IBM sequencer does not support this option.

## **MASTER SMPTE**

Sets the MIDI sequencer as the synchronization source. Synchronization data is sent in SMPTE format. The IBM sequencer does not support this option.

## **OFFSET time**

The SMPTE offset *time* in colon form (HOURS:MINUTES:SECONDS:FRAMES).

## **PORT MAPPER**

The MIDI mapper is the port receiving the MIDI messages. This command will fail if the MIDI mapper or a port it needs is being used by another sequence or client.

## **PORT NONE**

Disables the sending of MIDI messages. Currently this function is not supported by the IBM sequencer.

## **SLAVE FILE**

Sets the MIDI sequencer to use file data as the synchronization source. This is the default.

## **SLAVE MIDI**

Sets the MIDI sequencer to use incoming MIDI data as the synchronization source. The sequencer recognizes synchronization data with the MIDI format. The IBM sequencer does not support this option.

## **SLAVE NONE**

Sets the MIDI sequencer to ignore synchronization data.

## **SLAVE SMPTE**

Sets the MIDI sequencer to use incoming MIDI data for the synchronization source. The sequencer recognizes synchronization data with the SMPTE format. The IBM sequencer does not support this option.

## **TEMPO integer**

The tempo of the sequence according to the current time format. For a ppqn-based file, the **integer** is interpreted as beats per minute. For a SMPTE based file, the **integer** is interpreted as frames per second.

## **TIME FORMAT MILLISECONDS**

Sets time format to milliseconds. All position information is specified as milliseconds following this command. The sequence file sets the default format to ppqn or SMPTE. You can abbreviate milliseconds as **ms**.

## **TIME FORMAT MMTIME**

Sets the time format to MMTIME.

## **TIME FORMAT SONG POINTER**

Sets the time format to song-pointer (sixteenth notes). This can only be performed for a sequence of division type ppqn. Currently this function is not supported by the IBM sequencer.

## **TIME FORMAT SMPTE 24**

Sets the time format to SMPTE 24 frame rate. All position information is specified in SMPTE format following this command. The sequence file sets the default format to ppqn or SMPTE.

## **TIME FORMAT SMPTE 25**

Sets the time format to SMPTE 25 frame rate. All position information is specified in SMPTE format following this command. The

sequence file sets the default format to ppqn or SMPTE.

**TIME FORMAT SMPTE 30**

Sets the time format to SMPTE 30 frame rate. All position information is specified in SMPTE format following this command. The sequence file sets the default format to ppqn or SMPTE.

**TIME FORMAT SMPTE 30 DROP**

Sets the time format to SMPTE 30 drop frame rate. All position information is specified in SMPTE format following this command. The sequence file sets the default format to ppqn or SMPTE. Currently this function is not supported by the IBM sequencer.

**WAIT**

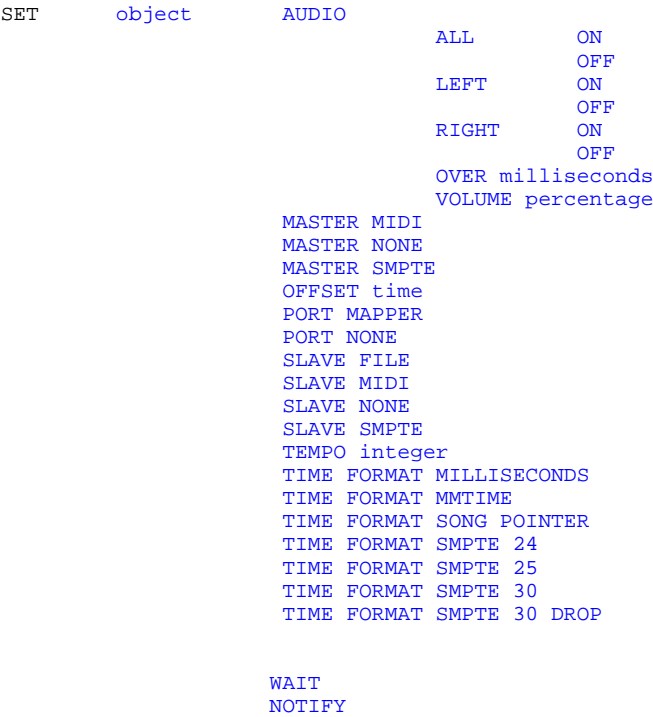
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# SET (MIDI Command) - Syntax Diagram



-----

# SET (MIDI Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## STATUS

---

### STATUS (MIDI Command) - Example

```
status sequencer ready wait
```

---

### STATUS (MIDI Command) - Purpose

The STATUS command obtains status information for the MIDI sequencer.

---

### STATUS (MIDI Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

### STATUS (MIDI Command) Keyword - CURRENT TRACK

**CURRENT TRACK**

Returns the current track number.

---

### STATUS (MIDI Command) Keyword - DIVISION TYPE

#### DIVISION TYPE

Returns the file division type. This can be one of: PPQN, SMPTE 24 frame, SMPTE 25 frame, SMPTE 30 frame, or SMPTE 30 drop frame. Use this information to determine the format of the MIDI file, and the meaning of tempo and position information.

-----

## STATUS (MIDI Command) Keyword - LENGTH

#### LENGTH

Returns the length of a sequence in the current time format. For ppqn files, this will be song pointer units. However, for SMPTE files, this will be in colon form (HOURS:MINUTES:SECONDS:FRAMES).

-----

## STATUS (MIDI Command) Keyword - LENGTH TRACK number

#### LENGTH TRACK number

Returns the length of a track in the current time format. For ppqn files, this will be song pointer units. However, for SMPTE files, this will be in colon form (HOURS:MINUTES:SECONDS:FRAMES).

-----

## STATUS (MIDI Command) Keyword - MASTER

#### MASTER

Returns **midi**, **none**, or **smp**, depending on the type of synchronization set.

-----

## STATUS (MIDI Command) Keyword - MEDIA PRESENT

#### MEDIA PRESENT

The sequencer returns TRUE.

-----

## STATUS (MIDI Command) Keyword - MODE

#### MODE

Returns the current mode of the device: **not ready**, **stopped**, **playing**, **seeking**, **recording**, **paused**, or **other**.

---

## STATUS (MIDI Command) Keyword - NUMBER OF TRACKS

### NUMBER OF TRACKS

Returns the number of tracks.

---

## STATUS (MIDI Command) Keyword - OFFSET

### OFFSET

Returns the offset of a SMPTE-based file. The offset is the beginning time of a SMPTE-based sequence. The MIDI sequencer driver returns the time in colon form (HOURS:MINUTES:SECONDS:FRAMES).

---

## STATUS (MIDI Command) Keyword - PORT

### PORT

Returns the MIDI port number assigned to the sequence. Currently this function is not supported by the IBM sequencer.

---

## STATUS (MIDI Command) Keyword - POSITION

### POSITION

Returns the current position of a sequence in the current time format. For ppqn files, this will be song pointer units. However, for SMPTE files, this will be in colon form (HOURS:MINUTES:SECONDS:FRAMES). Currently this function is not supported by the IBM sequencer.

---

## STATUS (MIDI Command) Keyword - POSITION TRACK number

### POSITION TRACK number

Returns the current position of the track specified by **number** in the current time format. For ppqn files, this will be song pointer units. However, for SMPTE files, this will be in colon form (HOURS:MINUTES:SECONDS:FRAMES).

---

## STATUS (MIDI Command) Keyword - READY



#### READY

Returns TRUE if the device is ready.

---

## STATUS (MIDI Command) Keyword - SLAVE

#### SLAVE

Returns **file**, **midi**, **none**, or **smpte** depending on the type of synchronization set.

---

## STATUS (MIDI Command) Keyword - START POSITION

#### START POSITION

Returns the starting position of the media or device element.

---

## STATUS (MIDI Command) Keyword - TEMPO

#### TEMPO

Returns the current tempo of a sequence in the current time format. For files with ppqn format, the tempo is in beats-per-minute. For files with SMPTE format, the tempo is in frames-per-second.

---

## STATUS (MIDI Command) Keyword - TIME FORMAT

#### TIME FORMAT

Returns the time format.

---

## STATUS (MIDI Command) Keyword - WAIT

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

---

# STATUS (MIDI Command) Keyword - NOTIFY

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## STATUS (MIDI Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### CURRENT TRACK

Returns the current track number.

### DIVISION TYPE

Returns the file division type. This can be one of: PPQN, SMPTE 24 frame, SMPTE 25 frame, SMPTE 30 frame, or SMPTE 30 drop frame. Use this information to determine the format of the MIDI file, and the meaning of tempo and position information.

### LENGTH

Returns the length of a sequence in the current time format. For ppqn files, this will be song pointer units. However, for SMPTE files, this will be in colon form (HOURS:MINUTES:SECONDS:FRAMES).

### LENGTH TRACK number

Returns the length of a track in the current time format. For ppqn files, this will be song pointer units. However, for SMPTE files, this will be in colon form (HOURS:MINUTES:SECONDS:FRAMES).

### MASTER

Returns **midi**, **none**, or **smp**te, depending on the type of synchronization set.

### MEDIA PRESENT

The sequencer returns TRUE.

### MODE

Returns the current mode of the device: **not ready**, **stopped**, **playing**, **seeking**, **recording**, **paused**, or **other**.

### NUMBER OF TRACKS

Returns the number of tracks.

### OFFSET

Returns the offset of a SMPTE-based file. The offset is the beginning time of a SMPTE-based sequence. The MIDI sequencer driver returns the time in colon form (HOURS:MINUTES:SECONDS:FRAMES).

### PORT

Returns the MIDI port number assigned to the sequence. Currently this function is not supported by the IBM sequencer.

### POSITION

Returns the current position of a sequence in the current time format. For ppqn files, this will be song pointer units. However, for SMPTE files, this will be in colon form (HOURS:MINUTES:SECONDS:FRAMES). Currently this function is not supported by the IBM sequencer.

### POSITION TRACK number

Returns the current position of the track specified by **number** in the current time format. For ppqn files, this will be song pointer units. However, for SMPTE files, this will be in colon form (HOURS:MINUTES:SECONDS:FRAMES).

### READY

Returns TRUE if the device is ready.

- SLAVE**  
Returns **file**, **midi**, **none**, or **smpte** depending on the type of synchronization set.
- START POSITION**  
Returns the starting position of the media or device element.
- TEMPO**  
Returns the current tempo of a sequence in the current time format. For files with ppqn format, the tempo is in beats-per-minute. For files with SMPTE format, the tempo is in frames-per-second.
- TIME FORMAT**  
Returns the time format.
- WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.
- NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

---

## STATUS (MIDI Command) - Syntax Diagram

STATUS	object	CURRENT TRACK	
		DIVISION TYPE	WAIT
		LENGTH	NOTIFY
		LENGTH TRACK number	
		MASTER	
		MEDIA PRESENT	
		MODE	
		NUMBER OF TRACKS	
		OFFSET	
		PORT	
		POSITION	
		POSITION TRACK number	
		READY	
		SLAVE	
		START POSITION	
		TEMPO	
		TIME FORMAT	



---

## STATUS (MIDI Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)
-

# Videodisc Player Commands

The videodisc device supports the following device-type specific commands and extensions to the following basic and required commands:

- CAPABILITY
- CONNECTOR
- CUE
- ESCAPE
- INFO
- PAUSE
- PLAY
- SEEK
- SET
- SPIN
- STATUS
- STEP

---

## CAPABILITY

---

### CAPABILITY (Videodisc Player Command) - Example

The following command returns FALSE.

```
capability videodisc can record wait
```

---

### CAPABILITY (Videodisc Player Command) - Purpose

The CAPABILITY command requests additional information about the capabilities of the device.

---

### CAPABILITY (Videodisc Player Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
-

## CAPABILITY (Videodisc Player Command) Keyword - CAN EJECT

### **CAN EJECT**

Returns TRUE if the device can eject the media.

---

## CAPABILITY (Videodisc Player Command) Keyword - CAN PLAY

### **CAN PLAY**

Returns TRUE.

---

## CAPABILITY (Videodisc Player Command) Keyword - CAN RECORD

### **CAN RECORD**

Returns FALSE. The device cannot record.

---

## CAPABILITY (Videodisc Player Command) Keyword - CAN REVERSE

### **CAN REVERSE**

Returns TRUE if the device can play in reverse.

---

## CAPABILITY (Videodisc Player Command) Keyword - CAN SAVE

### **CAN SAVE**

Returns FALSE. Videodisc players cannot save data.

---

## CAPABILITY (Videodisc Player Command) Keyword - CAN SETVOLUME

### CAN SETVOLUME

Returns TRUE if the device supports software control of volume level.

---

## CAPABILITY (Videodisc Player Command) Keyword - CAV

### CAV

When combined with other items, **CAV** specifies that the returned information applies to CAV formatted discs. This is the default.

---

## CAPABILITY (Videodisc Player Command) Keyword - CLV

### CLV

When combined with other items, **CLV** specifies that the returned information applies to CLV formatted discs.

---

## CAPABILITY (Videodisc Player Command) Keyword - COMPOUND DEVICE

### COMPOUND DEVICE

Returns FALSE. Videodisc players are simple devices.

---

## CAPABILITY (Videodisc Player Command) Keyword - DEVICE TYPE

### DEVICE TYPE

Returns **Videodisc**.

---

## CAPABILITY (Videodisc Player Command) Keyword - FAST

# PLAY RATE

## FAST PLAY RATE

Returns the fast play rate in the current speed format, either as a percentage or in frames per second. Returns 0 if the device cannot play fast.

-----

## CAPABILITY (Videodisc Player Command) Keyword - HAS AUDIO

### HAS AUDIO

Returns TRUE if the video device has audio.

-----

## CAPABILITY (Videodisc Player Command) Keyword - HAS VIDEO

### HAS VIDEO

Returns TRUE.

-----

## CAPABILITY (Videodisc Player Command) Keyword - MAXIMUM PLAY RATE

### MAXIMUM PLAY RATE

Returns the maximum play rate in the current speed format, either as a percentage or in frames per second.

-----

## CAPABILITY (Videodisc Player Command) Keyword - MESSAGE command

### MESSAGE command

Returns TRUE if the device supports the command specified by **command**. The **command** can be any string command such as OPEN, PLAY, and so on.

-----

## CAPABILITY (Videodisc Player Command) Keyword -

# MINIMUM PLAY RATE

**MINIMUM PLAY RATE**

Returns the minimum play rate in the current speed format, either as a percentage or in frames per second.

-----

# CAPABILITY (Videodisc Player Command) Keyword - NORMAL PLAY RATE

**NORMAL PLAY RATE**

Returns the normal rate of play in frames per second.

-----

# CAPABILITY (Videodisc Player Command) Keyword - PREROLL TIME

**PREROLL TIME**

Returns 0, indicating the preroll time is not bounded.

-----

# CAPABILITY (Videodisc Player Command) Keyword - PREROLL TYPE

**PREROLL TYPE**

Returns NOTIFIED.

-----

# CAPABILITY (Videodisc Player Command) Keyword - SLOW PLAY RATE

**SLOW PLAY RATE**

Returns the slow play rate in the current speed format, either as a percentage or in frames per second. Returns 0 if the device cannot play slow.

-----

# CAPABILITY (Videodisc Player Command) Keyword - USES



# FILES

## USES FILES

Returns FALSE. Videodisc players do not use files.

-----

## CAPABILITY (Videodisc Player Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

-----

## CAPABILITY (Videodisc Player Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CAPABILITY (Videodisc Player Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### CAN EJECT

Returns TRUE if the device can eject the media.

### CAN PLAY

Returns TRUE.

### CAN RECORD

Returns FALSE. The device cannot record.

### CAN REVERSE

Returns TRUE if the device can play in reverse.

### CAN SAVE

Returns FALSE. Videodisc players cannot save data.

### CAN SETVOLUME

Returns TRUE if the device supports software control of volume level.

**CAV**

When combined with other items, **CAV** specifies that the returned information applies to CAV formatted discs. This is the default.

**CLV**

When combined with other items, **CLV** specifies that the returned information applies to CLV formatted discs.

**COMPOUND DEVICE**

Returns FALSE. Videodisc players are simple devices.

**DEVICE TYPE**

Returns **Videodisc**.

**FAST PLAY RATE**

Returns the fast play rate in the current speed format, either as a percentage or in frames per second. Returns 0 if the device cannot play fast.

**HAS AUDIO**

Returns TRUE if the video device has audio.

**HAS VIDEO**

Returns TRUE.

**MAXIMUM PLAY RATE**

Returns the maximum play rate in the current speed format, either as a percentage or in frames per second.

**MESSAGE command**

Returns TRUE if the device supports the command specified by **command**. The **command** can be any string command such as OPEN, PLAY, and so on.

**MINIMUM PLAY RATE**

Returns the minimum play rate in the current speed format, either as a percentage or in frames per second.

**NORMAL PLAY RATE**

Returns the normal rate of play in frames per second.

**PREROLL TIME**

Returns 0, indicating the preroll time is not bounded.

**PREROLL TYPE**

Returns NOTIFIED.

**SLOW PLAY RATE**

Returns the slow play rate in the current speed format, either as a percentage or in frames per second. Returns 0 if the device cannot play slow.

**USES FILES**

Returns FALSE. Videodisc players do not use files.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CAPABILITY (Videodisc Player Command) - Syntax Diagram

CAPABILITY	<a href="#">object</a>	<a href="#">CAN EJECT</a>	
		<a href="#">CAN PLAY</a>	<a href="#">WAIT</a>
		<a href="#">CAN RECORD</a>	<a href="#">NOTIFY</a>
		<a href="#">CAN REVERSE</a>	
		<a href="#">CAN SAVE</a>	
		<a href="#">CAN SETVOLUME</a>	
		<a href="#">CAV</a>	

CLV  
COMPOUND\_DEVICE  
DEVICE\_TYPE  
FAST\_PLAY\_RATE  
HAS\_AUDIO  
HAS\_VIDEO  
MAXIMUM\_PLAY\_RATE  
MESSAGE\_command  
MINIMUM\_PLAY\_RATE  
NORMAL\_PLAY\_RATE  
PREROLL\_TIME  
PREROLL\_TYPE  
SLOW\_PLAY\_RATE  
USES\_FILES

Examples

---

## CAPABILITY (Videodisc Player Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## CONNECTOR

---

## CONNECTOR (Videodisc Player Command) - Example

```
connector videodisc enable type line out
```

---

## CONNECTOR (Videodisc Player Command) - Purpose

The CONNECTOR command enables, disables, or queries the status of connectors on a device.

---

## CONNECTOR (Videodisc Player Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## CONNECTOR (Videodisc Player Command) Keyword - ENABLE

**ENABLE**

Enables information flow through the indicated connector. Using this option requires that the NUMBER or TYPE keywords, or both also be specified.

-----

## CONNECTOR (Videodisc Player Command) Keyword - DISABLE

**DISABLE**

Disables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both also be specified.

-----

## CONNECTOR (Videodisc Player Command) Keyword - QUERY

**QUERY**

Queries the status of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Using this option requires that the NUMBER or TYPE keywords, or both also be specified.

-----

## CONNECTOR (Videodisc Player Command) Keyword - NUMBER connector\_number

**NUMBER connector\_number**

The connector number on which to perform the requested action. If the TYPE keyword is included, the connector number is interpreted as a relative offset within the specified connector type.

-----

# CONNECTOR (Videodisc Player Command) Keyword - TYPE connector\_type

## TYPE connector\_type

Indicates the type of connector to which the requested action applies. The following connector types are directly supported by this device:

line out

The line output connector. This connector is usually attached to the line in connector of another device such as a set of amplified speakers or other audio devices.

video out

The video output connector. This connector is usually attached to the video input of an external monitor or a video overlay board.

-----

# CONNECTOR (Videodisc Player Command) Keyword - WAIT

## WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

# CONNECTOR (Videodisc Player Command) Keyword - NOTIFY

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# CONNECTOR (Videodisc Player Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## ENABLE

Enables information flow through the indicated connector. Using this option requires that the NUMBER or TYPE keywords, or both also be specified.

## DISABLE

Disables information flow through the indicated connector. Use of this option requires that the NUMBER or TYPE keywords, or both

also be specified.

#### QUERY

Queries the status of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Using this option requires that the NUMBER or TYPE keywords, or both also be specified.

#### NUMBER connector\_number

The connector number on which to perform the requested action. If the TYPE keyword is included, the connector number is interpreted as a relative offset within the specified connector type.

#### TYPE connector\_type

Indicates the type of connector to which the requested action applies. The following connector types are directly supported by this device:

line out

The line output connector. This connector is usually attached to the line in connector of another device such as a set of amplified speakers or other audio devices.

video out

The video output connector. This connector is usually attached to the video input of an external monitor or a video overlay board.

#### WAIT

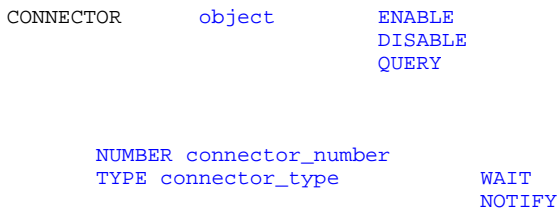
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CONNECTOR (Videodisc Player Command) - Syntax Diagram



### Examples

-----

## CONNECTOR (Videodisc Player Command) - Topics

Select an item:

- [Purpose](#)
- [Syntax Diagram](#)
- [Keywords](#)
- [Example](#)
- [Glossary](#)

---

## CUE

---

## CUE (Videodisc Player Command) - Example

```
cue videodisc input wait
```

---

## CUE (Videodisc Player Command) - Purpose

The CUE command prepares for playback or recording. The CUE command does not have to be issued prior to playback or recording; however, depending on the device, it might reduce the delay associated with the PLAY or RECORD command. The command fails if playing or recording is in progress.

The CUE command is not related to the [SETCUEPOINT](#) command.

---

## CUE (Videodisc Player Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CUE (Videodisc Player Command) Keyword - INPUT

### **INPUT**

Prepares the device for recording.

---

## CUE (Videodisc Player Command) Keyword - OUTPUT

**OUTPUT**

Prepares the device for playback. This is the default.

---

## CUE (Videodisc Player Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## CUE (Videodisc Player Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CUE (Videodisc Player Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**INPUT**

Prepares the device for recording.

**OUTPUT**

Prepares the device for playback. This is the default.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CUE (Videodisc Player Command) - Syntax Diagram



CUE                    [object](#)



## CUE (Videodisc Player Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

## ESCAPE

## ESCAPE (Videodisc Player Command) - Example

```
escape videodisc ?? wait
```

## ESCAPE (Videodisc Player Command) - Purpose

The ESCAPE command sends custom information to a device.

## ESCAPE (Videodisc Player Command) Keyword - object

- object**      Object associated with this media control interface command. The object can be one of the following:
- Device type
  - Device name
  - Filename

- Alias

-----

## ESCAPE (Videodisc Player Command) Keyword - string

### **string**

The custom information sent to the device.

-----

## ESCAPE (Videodisc Player Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## ESCAPE (Videodisc Player Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## ESCAPE (Videodisc Player Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### **string**

The custom information sent to the device.

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## ESCAPE (Videodisc Player Command) - Syntax Diagram

ESCAPE

object

string

WAIT  
NOTIFY

## Examples

---

# ESCAPE (Videodisc Player Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## INFO

---

# INFO (Videodisc Player Command) - Example

```
info videodisc product wait
```

---

## INFO (Videodisc Player Command) - Purpose

The INFO command fills a user-supplied buffer with information.

---

# INFO (Videodisc Player Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## INFO (Videodisc Player Command) Keyword - PRODUCT

### PRODUCT

Returns the product name of the device the peripheral is controlling.

-----

## INFO (Videodisc Player Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## INFO (Videodisc Player Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## INFO (Videodisc Player Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### PRODUCT

Returns the product name of the device the peripheral is controlling.

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# INFO (Videodisc Player Command) - Syntax Diagram

INFO      object      PRODUCT      WAIT  
NOTIFY



-----

# INFO (Videodisc Player Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

-----

# PAUSE

-----

# PAUSE (Videodisc Player Command) - Example

```
pause videodisc wait
```

-----

# PAUSE (Videodisc Player Command) - Purpose

The PAUSE command pauses playing. For CAV discs, it also freezes the video frame.

-----

# PAUSE (Videodisc Player Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## PAUSE (Videodisc Player Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## PAUSE (Videodisc Player Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## PAUSE (Videodisc Player Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## PAUSE (Videodisc Player Command) - Syntax Diagram

PAUSE

[object](#)

[WAIT](#)  
[NOTIFY](#)

## Examples

---

# PAUSE (Videodisc Player Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## PLAY

---

# PLAY (Videodisc Player Command) - Example

```
play videodisc fast notify
```

---

# PLAY (Videodisc Player Command) - Purpose

The PLAY command starts playing.

---

# PLAY (Videodisc Player Command) Keyword - object

## **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

## PLAY (Videodisc Player Command) Keyword - FROM pos

### **FROM pos**

Specifies the frame at which to start playing. If FROM is omitted, PLAY starts at the current position.

---

## PLAY (Videodisc Player Command) Keyword - TO pos

### **TO pos**

Specifies the frame at which to stop playing. If TO is omitted, PLAY stops at the end frame.

---

## PLAY (Videodisc Player Command) Keyword - FAST

### **FAST**

The device is to play faster than normal. To specify the speed more precisely, use the SPEED keyword. To determine the exact speed of a particular player, use the [STATUS](#) object SPEED command.

---

## PLAY (Videodisc Player Command) Keyword - REVERSE

### **REVERSE**

The play direction is backwards. This applies only to CAV discs.

---

## PLAY (Videodisc Player Command) Keyword - SLOW

### **SLOW**

The device is to play slower than normal.

---

## PLAY (Videodisc Player Command) Keyword - SCAN



#### SCAN

The play speed is as fast as possible, possibly with audio disabled. This applies only to CAV discs.

-----

## PLAY (Videodisc Player Command) Keyword - SPEED units

#### SPEED units

Play at the specified speed. Speed is set in units specified by [SET](#) object SPEED FORMAT. This applies only to CAV discs.

-----

## PLAY (Videodisc Player Command) Keyword - WAIT

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## PLAY (Videodisc Player Command) Keyword - NOTIFY

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## PLAY (Videodisc Player Command) - Keywords

#### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

#### FROM pos

Specifies the frame at which to start playing. If FROM is omitted, PLAY starts at the current position.

#### TO pos

Specifies the frame at which to stop playing. If TO is omitted, PLAY stops at the end frame.

#### FAST

The device is to play faster than normal. To specify the speed more precisely, use the SPEED keyword. To determine the exact speed of a particular player, use the [STATUS](#) object SPEED command.

#### REVERSE

The play direction is backwards. This applies only to CAV discs.

#### SLOW

The device is to play slower than normal.

**SCAN**  
The play speed is as fast as possible, possibly with audio disabled. This applies only to CAV discs.

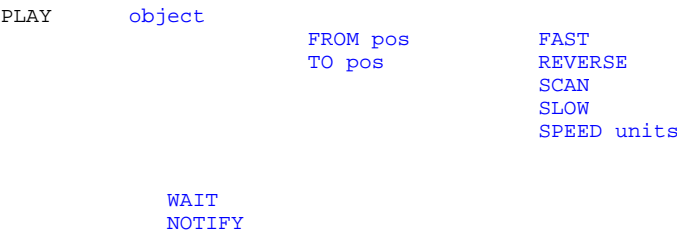
**SPEED units**  
Play at the specified speed. Speed is set in units specified by [SET](#) object SPEED FORMAT. This applies only to CAV discs.

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## PLAY (Videodisc Player Command) - Syntax Diagram



### Examples

---

## PLAY (Videodisc Player Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

## SEEK

---

## SEEK (Videodisc Player Command) - Example

```
seek videodisc01 to start wait
```

---

## SEEK (Videodisc Player Command) - Purpose

The SEEK command searches, using fast forward or fast reverse, with video and audio off.

---

## SEEK (Videodisc Player Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## SEEK (Videodisc Player Command) Keyword - REVERSE

### **REVERSE**

The seek direction on CAV discs is backwards. This modifier is invalid if TO is specified.

---

## SEEK (Videodisc Player Command) Keyword - TO pos

### **TO pos**

Specifies the final position to stop the seek. If TO is not specified, the seek continues until the end of the media is reached.

---

## SEEK (Videodisc Player Command) Keyword - TO START

### **TO START**

Seeks to the start of the disc.

---

## SEEK (Videodisc Player Command) Keyword - TO END

### TO END

Seeks to the end of the disc.

---

## SEEK (Videodisc Player Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## SEEK (Videodisc Player Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## SEEK (Videodisc Player Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### REVERSE

The seek direction on CAV discs is backwards. This modifier is invalid if TO is specified.

### TO pos

Specifies the final position to stop the seek. If TO is not specified, the seek continues until the end of the media is reached.

### TO START

Seeks to the start of the disc.

### TO END

Seeks to the end of the disc.

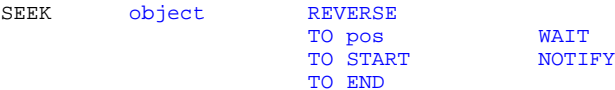
### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

SEEK (Videodisc Player Command) - Syntax Diagram



SEEK (Videodisc Player Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

SET

SET (Videodisc Player Command) - Example

```
set videodisc time format milliseconds wait
```

SET (Videodisc Player Command) - Purpose

The SET command sets the various control and attribute items.

## SET (Videodisc Player Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## SET (Videodisc Player Command) Keyword - AUDIO

### **AUDIO**

Specifies the audio attributes of the device context determined by the ALL, LEFT, and RIGHT keywords.

-----

## SET (Videodisc Player Command) Keyword - ALL

### **ALL**

Applies to both or all of the channels (default).

Specify ON or OFF with the ALL keyword.

-----

## SET (Videodisc Player Command) Keyword - ON

### **ON**

Enables audio.

-----

## SET (Videodisc Player Command) Keyword - OFF

### **OFF**

Disables audio.

-----

## SET (Videodisc Player Command) Keyword - LEFT

**LEFT**

Applies to the left channel.

Specify ON or OFF with the LEFT keyword.

---

## SET (Videodisc Player Command) Keyword - ON

**ON**

Enables output to the left audio channel.

---

## SET (Videodisc Player Command) Keyword - OFF

**OFF**

Disables output to the left audio channel.

---

## SET (Videodisc Player Command) Keyword - RIGHT

**RIGHT**

Applies to the right channel.

Specify ON or OFF with the RIGHT keyword.

---

## SET (Videodisc Player Command) Keyword - ON

**ON**

Enables output to the right audio channel.

---

## SET (Videodisc Player Command) Keyword - OFF

**OFF**

Disables output to the right audio channel.

---

## SET (Videodisc Player Command) Keyword - DISPLAY ON

**DISPLAY ON**  
Enables on-screen information display.

---

## SET (Videodisc Player Command) Keyword - DISPLAY OFF

**DISPLAY OFF**  
Disables on-screen information display.

---

## SET (Videodisc Player Command) Keyword - DOOR OPEN

**DOOR OPEN**  
Opens the door and ejects the tray, if possible.

---

## SET (Videodisc Player Command) Keyword - DOOR CLOSED

**DOOR CLOSED**  
Retracts the tray and closes the door, if possible.

---

## SET (Videodisc Player Command) Keyword - SPEED FORMAT PERCENTAGE

**SPEED FORMAT PERCENTAGE**  
Sets the speed format to percentage.

---

## SET (Videodisc Player Command) Keyword - SPEED FORMAT FPS



#### SPEED FORMAT FPS

The speed format to frames per second.

---

## SET (Videodisc Player Command) Keyword - TIME FORMAT MILLISECONDS

#### TIME FORMAT MILLISECONDS

Sets the position format to milliseconds. All position information is this format following this command. You can abbreviate milliseconds as **ms**.

---

## SET (Videodisc Player Command) Keyword - TIME FORMAT MMTIME

#### TIME FORMAT MMTIME

Sets the time position format to MMTIME.

---

## SET (Videodisc Player Command) Keyword - TIME FORMAT FRAMES

#### TIME FORMAT FRAMES

Sets the position format to frames. All position information is specified in frames following this command. When the device is opened, **frames** is the default mode for videodisc devices.

---

## SET (Videodisc Player Command) Keyword - TIME FORMAT HMS

#### TIME FORMAT HMS

Sets position format to *h:mm:ss*, where *h* is hours, *mm* is minutes, and *ss* is seconds. All position information is specified in this format following this command. The *h* input can be omitted if it is equal to 0, *mm* can be omitted if both *mm* and *h* equal 0.

---

## SET (Videodisc Player Command) Keyword - TIME FORMAT

# HMSF

## TIME FORMAT HMSF

Sets time format to hours, minutes, seconds, and frames. All position information is this format following this command.

-----

## SET (Videodisc Player Command) Keyword - VIDEO ON

### VIDEO ON

Enables video output.

-----

## SET (Videodisc Player Command) Keyword - VIDEO OFF

### VIDEO OFF

Disables video output.

-----

## SET (Videodisc Player Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## SET (Videodisc Player Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## SET (Videodisc Player Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type

- Device name
- Filename
- Alias

## AUDIO

Specifies the audio attributes of the device context determined by the ALL, LEFT, and RIGHT keywords.

### ALL

Applies to both or all of the channels (default).

Specify ON or OFF with the ALL keyword.

#### ON

Enables audio.

#### OFF

Disables audio.

### LEFT

Applies to the left channel.

Specify ON or OFF with the LEFT keyword.

#### ON

Enables output to the left audio channel.

#### OFF

Disables output to the left audio channel.

### RIGHT

Applies to the right channel.

Specify ON or OFF with the RIGHT keyword.

#### ON

Enables output to the right audio channel.

#### OFF

Disables output to the right audio channel.

## DISPLAY ON

Enables on-screen information display.

## DISPLAY OFF

Disables on-screen information display.

## DOOR OPEN

Opens the door and ejects the tray, if possible.

## DOOR CLOSED

Retracts the tray and closes the door, if possible.

## SPEED FORMAT PERCENTAGE

Sets the speed format to percentage.

## SPEED FORMAT FPS

The speed format to frames per second.

## TIME FORMAT MILLISECONDS

Sets the position format to milliseconds. All position information is this format following this command. You can abbreviate milliseconds as **ms**.

## TIME FORMAT MMTIME

Sets the time position format to MMTIME.

## TIME FORMAT FRAMES

Sets the position format to frames. All position information is specified in frames following this command. When the device is opened, **frames** is the default mode for videodisc devices.

## TIME FORMAT HMS

Sets position format to *h:mm:ss*, where *h* is hours, *mm* is minutes, and *ss* is seconds. All position information is specified in this format following this command. The *h* input can be omitted if it is equal to 0, *mm* can be omitted if both *mm* and *h* equal 0.

**TIME FORMAT HMSF**  
Sets time format to hours, minutes, seconds, and frames. All position information is this format following this command.

**VIDEO ON**  
Enables video output.

**VIDEO OFF**  
Disables video output.

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## SET (Videodisc Player Command) - Syntax Diagram

```
SET      object      AUDIO      ALL      ON
                                     OFF
                                     LEFT    ON
                                     RIGHT   OFF
                                     OFF
                                     ON
                                     OFF

        DISPLAY ON
        DISPLAY OFF
        DOOR OPEN
        DOOR CLOSED
        SPEED FORMAT PERCENTAGE
        SPEED FORMAT FPS
        TIME FORMAT FRAMES
        TIME FORMAT HMS
        TIME FORMAT HMSF
        TIME FORMAT MILLISECONDS
        TIME FORMAT MMTIME
        VIDEO ON
        VIDEO OFF

        WAIT
        NOTIFY
```



---

## SET (Videodisc Player Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)
-

# SPIN

---

## SPIN (Videodisc Player Command) - Example

```
spin videodisc up wait
```

---

## SPIN (Videodisc Player Command) - Purpose

The SPIN command starts the disc spinning or stops the disc from spinning.

---

## SPIN (Videodisc Player Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## SPIN (Videodisc Player Command) Keyword - DOWN

### **DOWN**

Stops the disc from spinning.

---

## SPIN (Videodisc Player Command) Keyword - UP

### **UP**

Starts the disc spinning.

---

# SPIN (Videodisc Player Command) Keyword - WAIT

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

# SPIN (Videodisc Player Command) Keyword - NOTIFY

**NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

# SPIN (Videodisc Player Command) - Keywords

**object**  
Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**DOWN**  
Stops the disc from spinning.

**UP**  
Starts the disc spinning.

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

# SPIN (Videodisc Player Command) - Syntax Diagram

SPIN      [object](#)      [DOWN](#)  
                                 [UP](#)      [WAIT](#)  
                                 [NOTIFY](#)



---

## SPIN (Videodisc Player Command) - Topics

Select an item:

[Purpose](#)

[Syntax Diagram](#)

[Keywords](#)

[Example](#)

[Glossary](#)

---

## STATUS

---

### STATUS (Videodisc Player Command) - Example

```
status videodisc media present wait
```

---

### STATUS (Videodisc Player Command) - Purpose

The STATUS command obtains status information for the device.

---

### STATUS (Videodisc Player Command) Keyword - object

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

### STATUS (Videodisc Player Command) Keyword - CURRENT TRACK

**CURRENT TRACK**

Returns the current chapter if applicable.

-----

## STATUS (Videodisc Player Command) Keyword - DISC SIZE

**DISC SIZE**

Returns either 8 or 12 to indicate the size of the loaded disc in inches.

-----

## STATUS (Videodisc Player Command) Keyword - FORWARD

**FORWARD**

Returns TRUE if the play direction is forward or if the device is not playing; FALSE if the play direction is backward.

-----

## STATUS (Videodisc Player Command) Keyword - LENGTH

**LENGTH**

Returns the total length of the segment.

-----

## STATUS (Videodisc Player Command) Keyword - LENGTH TRACK number

**LENGTH TRACK number**

Returns unsupported function.

-----

## STATUS (Videodisc Player Command) Keyword - MEDIA PRESENT

**MEDIA PRESENT**

Returns TRUE if the media is inserted in the device; otherwise, the return is FALSE.



---

## STATUS (Videodisc Player Command) Keyword - MEDIA TYPE

### MEDIA TYPE

Returns either **CAV**, **CLV**, or **other**, depending on the type of videodisc.

---

## STATUS (Videodisc Player Command) Keyword - MODE

### MODE

Returns **not ready**, **paused**, **playing**, **recording**, **seeking**, or **stopped** or **other**.

---

## STATUS (Videodisc Player Command) Keyword - NUMBER OF TRACKS

### NUMBER OF TRACKS

Returns the number of tracks on the media.

---

## STATUS (Videodisc Player Command) Keyword - POSITION

### POSITION

Returns the current position.

---

## STATUS (Videodisc Player Command) Keyword - POSITION TRACK number

### POSITION TRACK number

Returns unsupported function.

---

## STATUS (Videodisc Player Command) Keyword - READY

**READY**

Returns TRUE if the device is ready.

---

## STATUS (Videodisc Player Command) Keyword - SIDE

**SIDE**

Returns 1 or 2 to indicate which side of the disc is loaded.

---

## STATUS (Videodisc Player Command) Keyword - SPEED

**SPEED**

Returns the speed in the currently specified speed format.

---

## STATUS (Videodisc Player Command) Keyword - SPEED FORMAT

**SPEED FORMAT**

Returns the speed format.

---

## STATUS (Videodisc Player Command) Keyword - START POSITION

**START POSITION**

Returns the starting position of the media.

---

## STATUS (Videodisc Player Command) Keyword - TIME FORMAT

**TIME FORMAT**

Returns the time format.

---

## STATUS (Videodisc Player Command) Keyword - VOLUME

### VOLUME

Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

---

## STATUS (Videodisc Player Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

---

## STATUS (Videodisc Player Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## STATUS (Videodisc Player Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### CURRENT TRACK

Returns the current chapter if applicable.

### DISC SIZE

Returns either 8 or 12 to indicate the size of the loaded disc in inches.

### FORWARD

Returns TRUE if the play direction is forward or if the device is not playing; FALSE if the play direction is backward.

### LENGTH

Returns the total length of the segment.

### LENGTH TRACK number

Returns unsupported function.

**MEDIA PRESENT**

Returns TRUE if the media is inserted in the device; otherwise, the return is FALSE.

**MEDIA TYPE**

Returns either **CAV**, **CLV**, or **other**, depending on the type of videodisc.

**MODE**

Returns **not ready**, **paused**, **playing**, **recording**, **seeking**, or **stopped** or **other**.

**NUMBER OF TRACKS**

Returns the number of tracks on the media.

**POSITION**

Returns the current position.

**POSITION TRACK number**

Returns unsupported function.

**READY**

Returns TRUE if the device is ready.

**SIDE**

Returns 1 or 2 to indicate which side of the disc is loaded.

**SPEED**

Returns the speed in the currently specified speed format.

**SPEED FORMAT**

Returns the speed format.

**START POSITION**

Returns the starting position of the media.

**TIME FORMAT**

Returns the time format.

**VOLUME**

Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

-----

## STATUS (Videodisc Player Command) - Syntax Diagram

STATUS	object	CURRENT TRACK	
		DISC SIZE	WAIT
		FORWARD	NOTIFY
		LENGTH	
		LENGTH TRACK number	
		MEDIA PRESENT	
		MEDIA TYPE	
		MODE	
		NUMBER OF TRACKS	
		POSITION	
		POSITION TRACK number	
		READY	
		SIDE	

[SPEED](#)  
[SPEED FORMAT](#)  
[START POSITION](#)  
[TIME FORMAT](#)  
[VOLUME](#)

## Examples

---

# STATUS (Videodisc Player Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## STEP

---

# STEP (Videodisc Player Command) - Example

```
step videodisc reverse wait
```

---

# STEP (Videodisc Player Command) - Purpose

The STEP command steps the play one or more frames forward or backward. The default action is to step one time unit forward. This command applies only to CAV discs.

---

# STEP (Videodisc Player Command) Keyword - object

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name

- Filename
- Alias

-----

## STEP (Videodisc Player Command) Keyword - REVERSE

### REVERSE

Steps the frames in reverse. Only steps to I-frames.

-----

## STEP (Videodisc Player Command) Keyword - BY frames

### BY frames

Indicates the number of frames to step.

-----

## STEP (Videodisc Player Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## STEP (Videodisc Player Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## STEP (Videodisc Player Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### REVERSE

Steps the frames in reverse. Only steps to I-frames.

**BY frames**

Indicates the number of frames to step.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## STEP (Videodisc Player Command) - Syntax Diagram

STEP      [object](#)      [REVERSE](#)  
                                 [BY frames](#)      [WAIT](#)  
   [NOTIFY](#)



---

## STEP (Videodisc Player Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## Video Overlay Commands

The video overlay commands are the commands supported by the video overlay device for analog video. Information specific to the M-Motion Video Adapter/A, such as default values, is also provided.

The video overlay device for analog video supports the following device-type specific commands and extensions to the following basic and required commands:

- [CAPABILITY](#)
- [CAPTURE](#)
- [CONNECTOR](#)
- [FREEZE](#)
- [INFO](#)
- [LOAD](#)
- [OPEN](#)
- [PUT](#)
- [RESTORE](#)

- [SAVE](#)
- [SET](#)
- [STATUS](#)
- [UNFREEZE](#)
- [WHERE](#)
- [WINDOW](#)

-----

## CAPABILITY

-----

### CAPABILITY (Video Overlay Command) - Example

```
capability videooverlay can distort wait
```

-----

### CAPABILITY (Video Overlay Command) - Purpose

The CAPABILITY command requests information about the capabilities of the video overlay device driver.

-----

### CAPABILITY (Video Overlay Command) Keyword - object

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

### CAPABILITY (Video Overlay Command) Keyword - CAN DISTORT

#### **CAN DISTORT**

Returns TRUE if the device can stretch and display incoming video independently in horizontal and vertical dimensions.

-----



# CAPABILITY (Video Overlay Command) Keyword - CAN EJECT

**CAN EJECT**  
Returns FALSE.

-----

# CAPABILITY (Video Overlay Command) Keyword - CAN FREEZE

**CAN FREEZE**  
Returns TRUE if the device can freeze data in the frame buffer.

-----

# CAPABILITY (Video Overlay Command) Keyword - CAN LOCKEJECT

**CAN LOCKEJECT**  
Returns FALSE.

-----

# CAPABILITY (Video Overlay Command) Keyword - CAN OVERLAY GRAPHICS

**CAN OVERLAY GRAPHICS**  
Returns TRUE if the device can display graphics over video.

-----

# CAPABILITY (Video Overlay Command) Keyword - CAN PLAY

**CAN PLAY**  
Returns FALSE.

-----

# CAPABILITY (Video Overlay Command) Keyword - CAN RECORD

**CAN RECORD**  
Returns FALSE. The overlay device cannot record. However, the external device connected to the overlay device might be able to record.

# CAPABILITY (Video Overlay Command) Keyword - CAN SAVE

**CAN SAVE**  
Returns TRUE if the device can save video still image frames to a file.

# CAPABILITY (Video Overlay Command) Keyword - CAN SETVOLUME

**CAN SETVOLUME**  
Returns FALSE. Video overlay devices do not control audio.

# CAPABILITY (Video Overlay Command) Keyword - CAN STRETCH

**CAN STRETCH**  
Returns TRUE if the device can stretch or shrink video to fill a given display rectangle.

# CAPABILITY (Video Overlay Command) Keyword - COMPOUND DEVICE

**COMPOUND DEVICE**  
Returns TRUE.

## CAPABILITY (Video Overlay Command) Keyword - DEVICE TYPE

**DEVICE TYPE**  
Returns **OVERLAY**.

---

## CAPABILITY (Video Overlay Command) Keyword - HAS AUDIO

**HAS AUDIO**  
Returns **FALSE**. Control of audio mixing on video overlay hardware is performed through the amp-mixer device.

---

## CAPABILITY (Video Overlay Command) Keyword - HAS IMAGE

**HAS IMAGE**  
Returns **TRUE** if the device supports still image functions.

---

## CAPABILITY (Video Overlay Command) Keyword - HAS VIDEO

**HAS VIDEO**  
Returns **TRUE**.

---

## CAPABILITY (Video Overlay Command) Keyword - HORIZONTAL IMAGE EXTENT

**HORIZONTAL IMAGE EXTENT**  
Returns the maximum horizontal (X) extent for still image capture.

**M-Motion specific:** Returns **640**.

---

## CAPABILITY (Video Overlay Command) Keyword - HORIZONTAL SOURCE EXTENT

### HORIZONTAL SOURCE EXTENT

Returns the maximum horizontal (X) extent for the video source.

**M-Motion specific:** Returns **706** for both NTSC and PAL video.

---

## CAPABILITY (Video Overlay Command) Keyword - MESSAGE command

### MESSAGE command

Returns TRUE if the overlay device supports the command specified by **command**. The **command** can be any string command such as OPEN, PLAY, and so on.

---

## CAPABILITY (Video Overlay Command) Keyword - PREROLL TIME

### PREROLL TIME

Returns 0.

---

## CAPABILITY (Video Overlay Command) Keyword - PREROLL TYPE

### PREROLL TYPE

Returns the preroll characteristics of the device. Returns NONE.

---

## CAPABILITY (Video Overlay Command) Keyword - USES FILES

### USES FILES

Returns TRUE if the device accepts file names for loading and saving images.

---

## CAPABILITY (Video Overlay Command) Keyword - VERTICAL IMAGE EXTENT

### VERTICAL IMAGE EXTENT

Returns the maximum vertical (Y) extent for still image capture.

**M-Motion specific:** Returns **480**.

---

## CAPABILITY (Video Overlay Command) Keyword - VERTICAL SOURCE EXTENT

### VERTICAL SOURCE EXTENT

Returns the maximum vertical (Y) extent for the video source.

**M-Motion specific:** Returns **484** for NTSC video or **564** for PAL video.

---

## CAPABILITY (Video Overlay Command) Keyword - WINDOWS

### WINDOWS

Returns an integer for the maximum number of windows that the device can support concurrently.

**M-Motion specific:** Returns **10**.

---

## CAPABILITY (Video Overlay Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

---

## CAPABILITY (Video Overlay Command) Keyword - NOTIFY

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# CAPABILITY (Video Overlay Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## CAN DISTORT

Returns TRUE if the device can stretch and display incoming video independently in horizontal and vertical dimensions.

## CAN EJECT

Returns FALSE.

## CAN FREEZE

Returns TRUE if the device can freeze data in the frame buffer.

## CAN LOCKEJECT

Returns FALSE.

## CAN OVERLAY GRAPHICS

Returns TRUE if the device can display graphics over video.

## CAN PLAY

Returns FALSE.

## CAN RECORD

Returns FALSE. The overlay device cannot record. However, the external device connected to the overlay device might be able to record.

## CAN SAVE

Returns TRUE if the device can save video still image frames to a file.

## CAN SETVOLUME

Returns FALSE. Video overlay devices do not control audio.

## CAN STRETCH

Returns TRUE if the device can stretch or shrink video to fill a given display rectangle.

## COMPOUND DEVICE

Returns TRUE.

## DEVICE TYPE

Returns **OVERLAY**.

## HAS AUDIO

Returns FALSE. Control of audio mixing on video overlay hardware is performed through the amp-mixer device.

## HAS IMAGE

Returns TRUE if the device supports still image functions.

## HAS VIDEO

Returns TRUE.

## HORIZONTAL IMAGE EXTENT

Returns the maximum horizontal (X) extent for still image capture.

**M-Motion specific:** Returns **640**.

**HORIZONTAL SOURCE EXTENT**

Returns the maximum horizontal (X) extent for the video source.

**M-Motion specific:** Returns **706** for both NTSC and PAL video.

**MESSAGE command**

Returns TRUE if the overlay device supports the command specified by **command**. The **command** can be any string command such as OPEN, PLAY, and so on.

**PREROLL TIME**

Returns 0.

**PREROLL TYPE**

Returns the preroll characteristics of the device. Returns NONE.

**USES FILES**

Returns TRUE if the device accepts file names for loading and saving images.

**VERTICAL IMAGE EXTENT**

Returns the maximum vertical (Y) extent for still image capture.

**M-Motion specific:** Returns **480**.

**VERTICAL SOURCE EXTENT**

Returns the maximum vertical (Y) extent for the video source.

**M-Motion specific:** Returns **484** for NTSC video or **564** for PAL video.

**WINDOWS**

Returns an integer for the maximum number of windows that the device can support concurrently.

**M-Motion specific:** Returns **10**.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete an MM\_MCINOTIFY message is sent to the application window procedure.

-----

CAPABILITY (Video Overlay Command) - Syntax Diagram

CAPABILITY	object	CAN DISTORT	
		CAN EJECT	WAIT
		CAN FREEZE	NOTIFY
		CAN LOCKEJECT	
		CAN OVERLAY GRAPHICS	
		CAN PLAY	
		CAN RECORD	
		CAN SAVE	
		CAN SETVOLUME	
		CAN STRETCH	
		COMPOUND DEVICE	
		DEVICE TYPE	
		HAS AUDIO	
		HAS IMAGE	
		HAS VIDEO	
		HORIZONTAL IMAGE EXTENT	
		HORIZONTAL SOURCE EXTENT	
		MESSAGE command	
		PREROLL TIME	
		PREROLL TYPE	
		USES FILES	
		VERTICAL IMAGE EXTENT	
		VERTICAL SOURCE EXTENT	

Examples

---

## CAPABILITY (Video Overlay Command) - Topics

Select an item:

[Purpose](#)

[Syntax Diagram](#)

[Keywords](#)

[Example](#)

[Glossary](#)

---

## CAPTURE

---

## CAPTURE (Video Overlay Command) - Example

```
capture videooverlay at 100 100 260 220 wait
```

---

## CAPTURE (Video Overlay Command) - Purpose

The CAPTURE command captures the current video image. This does not cause the image or bit map to be saved; the application must subsequently issue a SAVE command to save the device element. The device will freeze motion temporarily if needed to capture the image and put the image into the image device element. Repeated capture operations will overwrite the image contained in this temporary space. The device will wait for a SAVE command to transfer the information to a file, or MCI\_GETIMAGEBUFFER to supply an application with a copy of the image buffer.

**Note:** If no rectangle is specified, the entire rectangle is captured.

---

## CAPTURE (Video Overlay Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:



- Device type
- Device name
- Filename
- Alias

-----

## CAPTURE (Video Overlay Command) Keyword - AT rect

### AT rect

Specifies a rectangle relative to the window origin in device coordinates. The rectangle is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner. Only the video in that subregion will be captured.

-----

## CAPTURE (Video Overlay Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## CAPTURE (Video Overlay Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CAPTURE (Video Overlay Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### AT rect

Specifies a rectangle relative to the window origin in device coordinates. The rectangle is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner. Only the video in that subregion will be captured.

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CAPTURE (Video Overlay Command) - Syntax Diagram

CAPTURE      object      AT rect      WAIT  
NOTIFY



---

## CAPTURE (Video Overlay Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

## CONNECTOR

---

## CONNECTOR (Video Overlay Command) - Example

```
connector videooverlay query number 2 wait
```

---

## CONNECTOR (Video Overlay Command) - Purpose

The CONNECTOR command enables, disables, or queries the status of connector on a device.

---

## CONNECTOR (Video Overlay Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## CONNECTOR (Video Overlay Command) Keyword - ENABLE

**ENABLE**

Enables information flow through the indicated connector. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

-----

## CONNECTOR (Video Overlay Command) Keyword - DISABLE

**DISABLE**

Disables information flow through the indicated connector. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

-----

## CONNECTOR (Video Overlay Command) Keyword - QUERY

**QUERY**

Queries the state of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

-----

## CONNECTOR (Video Overlay Command) Keyword - NUMBER connector\_number

**NUMBER connector\_number**

Indicates the connector number on which to perform the requested action. If this item is omitted, then the first connector is assumed. If the TYPE item is included, then the connector number is interpreted as a relative offset within the specified connector type.

-----

## CONNECTOR (Video Overlay Command) Keyword - TYPE

# connector\_type

## TYPE connector\_type

Indicates the type of connector to which the requested action applies. The connector types are defined by each device.

-----

# CONNECTOR (Video Overlay Command) Keyword - WAIT

## WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

# CONNECTOR (Video Overlay Command) Keyword - NOTIFY

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# CONNECTOR (Video Overlay Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## ENABLE

Enables information flow through the indicated connector. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

## DISABLE

Disables information flow through the indicated connector. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

## QUERY

Queries the state of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

## NUMBER connector\_number

Indicates the connector number on which to perform the requested action. If this item is omitted, then the first connector is assumed. If the TYPE item is included, then the connector number is interpreted as a relative offset within the specified connector type.

## TYPE connector\_type

Indicates the type of connector to which the requested action applies. The connector types are defined by each device.

## WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the

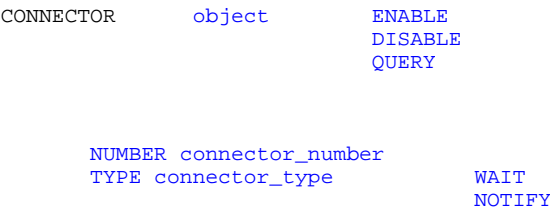
application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# CONNECTOR (Video Overlay Command) - Syntax Diagram



---

# CONNECTOR (Video Overlay Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

# FREEZE

---

# FREEZE (Video Overlay Command) - Example

```
freeze videooverlay at 100 100 260 220 wait
```

---

# FREEZE (Video Overlay Command) - Purpose

The FREEZE command stops updating the video buffer by the video source. Supported only if **can freeze** returns TRUE.

---

## FREEZE (Video Overlay Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## FREEZE (Video Overlay Command) Keyword - AT rect

### **AT rect**

Specifies a rectangle relative to the window origin in device coordinates. The rectangle array is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner. Only the video in that subregion will be frozen.

---

## FREEZE (Video Overlay Command) Keyword - OUTSIDE rect

### **OUTSIDE rect**

The area outside the specified rectangle is to be affected.

---

## FREEZE (Video Overlay Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## FREEZE (Video Overlay Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## FREEZE (Video Overlay Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### **AT rect**

Specifies a rectangle relative to the window origin in device coordinates. The rectangle array is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner. Only the video in that subregion will be frozen.

### **OUTSIDE rect**

The area outside the specified rectangle is to be affected.

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## FREEZE (Video Overlay Command) - Syntax Diagram

```
FREEZE      object      AT rect
              OUTSIDE rect      WAIT
                               NOTIFY
```

**Examples**

---

## FREEZE (Video Overlay Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

# INFO

---

## INFO (Video Overlay Command) - Example

```
info videooverlay product wait
```

---

## INFO (Video Overlay Command) - Purpose

The INFO command returns string information from the device driver.

---

## INFO (Video Overlay Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## INFO (Video Overlay Command) Keyword - FILE

### **FILE**

Returns the name of the current element.

---

## INFO (Video Overlay Command) Keyword - IMAGE

### **IMAGE**

Specifies an optional keyword indicating image file.

---



## INFO (Video Overlay Command) Keyword - PRODUCT

### **PRODUCT**

Returns the product name and model of the hardware used for the video overlay device.

-----

## INFO (Video Overlay Command) Keyword - WINDOW TEXT

### **WINDOW TEXT**

Returns the caption of the video overlay window.

-----

## INFO (Video Overlay Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## INFO (Video Overlay Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## INFO (Video Overlay Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### **FILE**

Returns the name of the current element.

### **IMAGE**

Specifies an optional keyword indicating image file.

**PRODUCT**

Returns the product name and model of the hardware used for the video overlay device.

**WINDOW TEXT**

Returns the caption of the video overlay window.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

INFO (Video Overlay Command) - Syntax Diagram



INFO (Video Overlay Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

LOAD

LOAD (Video Overlay Command) - Example

```
load videooverlay picture.vid wait
```

---

## LOAD (Video Overlay Command) - Purpose

The LOAD command loads a new device element (file) into an already open device context and overwrites any image currently stored there. It can be displayed using the RESTORE command. The file will be opened, accessed, and closed on this command. If the format of the image file is not recognizable as either a device specific file format or a format supported by MMIO, the load will fail.

---

## LOAD (Video Overlay Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## LOAD (Video Overlay Command) Keyword - filename

### **filename**

Specifies the file name to load.

---

## LOAD (Video Overlay Command) Keyword - IMAGE

### **IMAGE**

Specifies an optional keyword indicating an image file.

---

## LOAD (Video Overlay Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## LOAD (Video Overlay Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## LOAD (Video Overlay Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**filename**

Specifies the file name to load.

**IMAGE**

Specifies an optional keyword indicating an image file.

**WAIT**

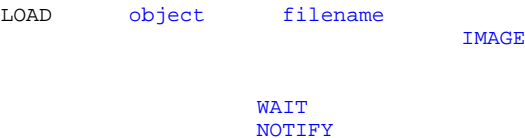
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## LOAD (Video Overlay Command) - Syntax Diagram



---

## LOAD (Video Overlay Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)

---

## OPEN

---

### OPEN (Video Overlay Command) - Example

```
open videooverlay alias vd shareable wait
```

---

### OPEN (Video Overlay Command) - Purpose

The OPEN command opens the video overlay device.

---

### OPEN (Video Overlay Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

### OPEN (Video Overlay Command) Keyword - ALIAS device\_alias

**ALIAS device\_alias**

Specifies an alternate name for the device. If specified, it must also be used for subsequent references.

---

### OPEN (Video Overlay Command) Keyword - DOSQUEUE

**DOSQUEUE**

If a device instance is opened with the DOSQUEUE keyword specified, window handles that are passed in for the instance will be treated as OS/2 Control Program queue handles.

---

## OPEN (Video Overlay Command) Keyword - PARENT hwnd

**PARENT hwnd**

Specifies the window handle of the parent window. If specified, it is used as the parent window of the digital video device default window.

---

## OPEN (Video Overlay Command) Keyword - READONLY

**READONLY**

Specifies that the file is to be opened in read-only mode.

---

## OPEN (Video Overlay Command) Keyword - SHAREABLE

**SHAREABLE**

Initializes the device as shareable. Specifying SHAREABLE makes the resources of the device available to other device contexts. If SHAREABLE is not specified on OPEN, the resource will be exclusively acquired when the device is opened.

---

## OPEN (Video Overlay Command) Keyword - TYPE device\_type

**TYPE device\_type**

Specifies the compound device used to control a device element. As an alternative to TYPE, the media control interface can use the extended attributes or file extensions associated with the file to select the controlling device.

---

## OPEN (Video Overlay Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

# OPEN (Video Overlay Command) Keyword - NOTIFY

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# OPEN (Video Overlay Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## ALIAS device\_alias

Specifies an alternate name for the device. If specified, it must also be used for subsequent references.

## DOSQUEUE

If a device instance is opened with the DOSQUEUE keyword specified, window handles that are passed in for the instance will be treated as OS/2 Control Program queue handles.

## PARENT hwnd

Specifies the window handle of the parent window. If specified, it is used as the parent window of the digital video device default window.

## READONLY

Specifies that the file is to be opened in read-only mode.

## SHAREABLE

Initializes the device as shareable. Specifying SHAREABLE makes the resources of the device available to other device contexts. If SHAREABLE is not specified on OPEN, the resource will be exclusively acquired when the device is opened.

## TYPE device\_type

Specifies the compound device used to control a device element. As an alternative to TYPE, the media control interface can use the extended attributes or file extensions associated with the file to select the controlling device.

## WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# OPEN (Video Overlay Command) - Syntax Diagram

OPEN

object

ALIAS device\_alias  
DOSQUEUE  
PARENT hwnd  
READONLY  
SHAREABLE  
TYPE device\_type

WAIT  
NOTIFY

Examples

-----

OPEN (Video Overlay Command) - Topics

- Select an item:
- [Purpose](#)
- [Syntax Diagram](#)
- [Keywords](#)
- [Example](#)
- [Glossary](#)

-----

PUT

-----

PUT (Video Overlay Command) - Example

put videooverlay source wait

-----

-----

PUT (Video Overlay Command) - Purpose

The PUT command sets the source and destination rectangles for the video and sets the size and position of the video window.

**Warning:** Setting the source rectangle smaller than the destination rectangle size might result in unpredictable video remnants appearing in the video window. Sizing the video window larger than the current source rectangle size might also produce this effect.

-----

-----

PUT (Video Overlay Command) Keyword - object



**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## PUT (Video Overlay Command) Keyword - SOURCE

**SOURCE**

Sets the source rectangle to the default size and position. The source rectangle specifies the portion of the video source which will be displayed and is relative to the lower-left corner of the video source.

-----

## PUT (Video Overlay Command) Keyword - DESTINATION

**DESTINATION**

Sets the default destination rectangle to the size of the video window. Therefore, the entire video window displays video. This destination rectangle will automatically be adjusted as the window is sized.

-----

## PUT (Video Overlay Command) Keyword - SOURCE DESTINATION

**SOURCE DESTINATION**

Sets both the source and destination rectangles to their respective defaults.

-----

## PUT (Video Overlay Command) Keyword - SOURCE AT rect

**SOURCE AT rect**

The source clipping rectangle specifies the portion of the source video which will be displayed. The rectangle is relative to the lower-left corner of the video source.

-----

## PUT (Video Overlay Command) Keyword - DESTINATION AT rect

#### DESTINATION AT rect

The destination rectangle specifies where in the video window that video will be displayed. All areas within the video window that are outside the destination rectangle will be frozen.

The rectangle is relative to the window origin and is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner.

-----

## PUT (Video Overlay Command) Keyword - WINDOW AT rect

#### WINDOW AT rect

Moves and/or sizes the default video window by specifying a valid rectangle and the following options:

**Note:** The MOVE and SIZE keywords can both be specified, in which case, the default video window is moved and sized at the same time.

-----

## PUT (Video Overlay Command) Keyword - MOVE

#### MOVE

Moves the default video window to the X1 Y1 coordinates specified in the rectangle. Window coordinates are relative to the parent window.

##### Notes:

- (1) All coordinates of the rectangle (X1 Y1 X2 Y2) must be specified, but X2 Y2 are ignored if the **size** option is not specified.
  - (2) This option will not affect an application-supplied alternate video window.
- 

## PUT (Video Overlay Command) Keyword - SIZE

#### SIZE

Sizes the default video window to the difference of the coordinates  $((X2 - X1) + 1)$  and  $((Y2 - Y1) + 1)$ . All coordinates of the rectangle (X1 Y1 X2 Y2) must be specified.

**Note:** This option will not affect an application-supplied alternate video window.

-----

## PUT (Video Overlay Command) Keyword - WAIT

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

# PUT (Video Overlay Command) Keyword - NOTIFY

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

-----

# PUT (Video Overlay Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## SOURCE

Sets the source rectangle to the default size and position. The source rectangle specifies the portion of the video source which will be displayed and is relative to the lower-left corner of the video source.

## DESTINATION

Sets the default destination rectangle to the size of the video window. Therefore, the entire video window displays video. This destination rectangle will automatically be adjusted as the window is sized.

## SOURCE DESTINATION

Sets both the source and destination rectangles to their respective defaults.

## SOURCE AT rect

The source clipping rectangle specifies the portion of the source video which will be displayed. The rectangle is relative to the lower-left corner of the video source.

## DESTINATION AT rect

The destination rectangle specifies where in the video window that video will be displayed. All areas within the video window that are outside the destination rectangle will be frozen.

The rectangle is relative to the window origin and is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner.

## WINDOW AT rect

Moves and/or sizes the default video window by specifying a valid rectangle and the following options:

**Note:** The MOVE and SIZE keywords can both be specified, in which case, the default video window is moved and sized at the same time.

## MOVE

Moves the default video window to the X1 Y1 coordinates specified in the rectangle. Window coordinates are relative to the parent window.

### Notes:

(1) All coordinates of the rectangle (X1 Y1 X2 Y2) must be specified, but X2 Y2 are ignored if the **size** option is not specified.

(2) This option will not affect an application-supplied alternate video window.

## SIZE

Sizes the default video window to the difference of the coordinates  $((X2 - X1) + 1)$  and  $((Y2 - Y1) + 1)$ . All coordinates of the rectangle (X1 Y1 X2 Y2) must be specified.

**Note:** This option will not affect an application-supplied alternate video window.

**WAIT**

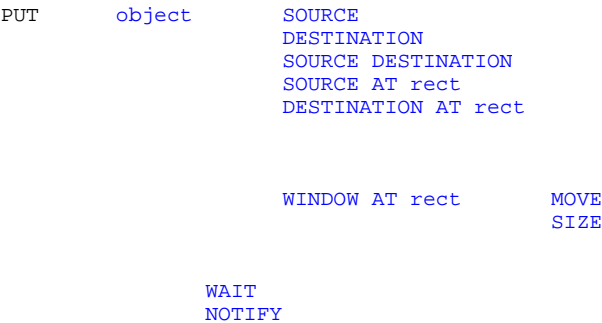
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## PUT (Video Overlay Command) - Syntax Diagram



-----

## PUT (Video Overlay Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

-----

## RESTORE

-----

## RESTORE (Video Overlay Command) - Example

```
restore videooverlay destination at 100 100 200 200 wait
```

---

## RESTORE (Video Overlay Command) - Purpose

The RESTORE command restores the video image from the currently loaded bitmap or image. The device transfers the image from the device element buffer to the display surface. To ensure that the image is displayed, the device automatically performs a FREEZE operation, if necessary, on the area covered by the image.

---

## RESTORE (Video Overlay Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## RESTORE (Video Overlay Command) Keyword - DESTINATION AT rect

### **DESTINATION AT rect**

Specifies the window subregion where the image is to be restored. The rectangle is relative to the window origin in device coordinates and is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner.

---

## RESTORE (Video Overlay Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## RESTORE (Video Overlay Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## RESTORE (Video Overlay Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### DESTINATION AT rect

Specifies the window subregion where the image is to be restored. The rectangle is relative to the window origin in device coordinates and is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner.

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## RESTORE (Video Overlay Command) - Syntax Diagram

```
RESTORE      object      DESTINATION AT rect

              WAIT
              NOTIFY
```

Examples

---

## RESTORE (Video Overlay Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## SAVE

---

## SAVE (Video Overlay Command) - Example

```
save videooverlay pic.vid wait
```

---

## SAVE (Video Overlay Command) - Purpose

The SAVE command saves the current image. The device will transfer the image in the image device element to a file, converting where possible, to support the current settings. For example, FileFormat, Quality, BitsPerPel and PelFormat.

---

## SAVE (Video Overlay Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## SAVE (Video Overlay Command) Keyword - filename

### **filename**

Specifies the destination path and filename.

---

## SAVE (Video Overlay Command) Keyword - IMAGE

### **IMAGE**

Specifies an optional keyword.

---

## SAVE (Video Overlay Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

# SAVE (Video Overlay Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# SAVE (Video Overlay Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**filename**

Specifies the destination path and filename.

**IMAGE**

Specifies an optional keyword.

**WAIT**

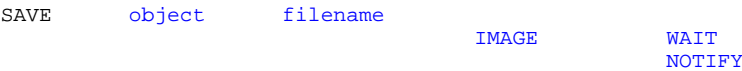
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# SAVE (Video Overlay Command) - Syntax Diagram





---

## SAVE (Video Overlay Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## SET

---

## SET (Video Overlay Command) - Example

```
set videooverlay greyscale on wait
```

---

## SET (Video Overlay Command) - Purpose

The SET command sets the various control and attribute items.

---

## SET (Video Overlay Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## SET (Video Overlay Command) Keyword - BRIGHTNESS level

**BRIGHTNESS level**

Sets the brightness to the specified level (0-100).

**M-Motion specific:** The default value is **80**.

---

## SET (Video Overlay Command) Keyword - CONTRAST level

**CONTRAST level**

Sets the contrast to the specified level (0-100).

**M-Motion specific:** The default value is **90**.

---

## SET (Video Overlay Command) Keyword - GREYSCALE ON

**GREYSCALE ON**

Enables greyscale. Video is displayed in black and white.

---

## SET (Video Overlay Command) Keyword - GREYSCALE OFF

**GREYSCALE OFF**

Disables greyscale. Video is displayed in color using the current settings.

**M-Motion specific:** The default is **off**.

---

## SET (Video Overlay Command) Keyword - HUE level

**HUE level**

Sets the hue to the specified level (0 - 100). A value of 50 specifies neutral hue, which is the default. Hue is also referred to as "tint."

**M-Motion specific:** The default is **50**.

---

## SET (Video Overlay Command) Keyword - IMAGE BITSPERPEL count

**IMAGE BITSPERPEL count**

Sets the number of bits per pixel for saving bit maps.

**M-Motion specific:** The default value is **12**.

---

## SET (Video Overlay Command) Keyword - IMAGE PELFORMAT type

### IMAGE PELFORMAT type

Sets the pel format or color encoding method for saving bit maps and images specified by the four-character code (FOURCC), such as yuvb or rgbb.

**M-Motion specific:** The default is **yuvb**.

---

## SET (Video Overlay Command) Keyword - IMAGE FILE FORMAT format

### IMAGE FILE FORMAT format

Sets the specific image file format in which the image capture is to be stored (when "saved"). This format must be specified by a four-character code (FOURCC), for example, MMOT or OS13, and must be one of the currently supported and installed MMIO image file formats, or the device-specific format. This does not effect the loading or restoring of images. It overwrites any previous file-format value, such as that obtained through a LOAD operation.

**M-Motion specific:** The default is **MMOT**.

---

## SET (Video Overlay Command) Keyword - IMAGE COMPRESSION type

### IMAGE COMPRESSION type

Sets the compression type to be used for saving images if possible. Possible values for **type** include: **none**.

**M-Motion specific:** The default is **none**.

The following **type** values are not supported:

- pic9
  - pic16
  - jpeg9
  - jpegn
  - rle4
  - rle8
- 

## SET (Video Overlay Command) Keyword - IMAGE QUALITY

# level

## IMAGE QUALITY level

Sets the specified still image quality level. Used for optimizing the auto-selection of compression type when saving to file formats.

high	Indicates photo-like image quality.
med	Indicates that the image has moderate complexity or quality.
low	Indicates a lower color or complexity image.

**M-Motion specific:** The default is **high**.

-----

## SET (Video Overlay Command) Keyword - SATURATION level

### SATURATION level

Sets the saturation to the specified level (0-100). Saturation is also referred to as "color."

**M-Motion specific:** The default value is **65**.

-----

## SET (Video Overlay Command) Keyword - SHARPNESS level

### SHARPNESS level

Sets the sharpness to the specified level (0-100).

**M-Motion specific:** The default value is **80**.

-----

## SET (Video Overlay Command) Keyword - VIDEO ON

### VIDEO ON

Enables video output.

-----

## SET (Video Overlay Command) Keyword - VIDEO OFF

### VIDEO OFF

Disables video output. The video window will be black. The following operations have no effect if video is set off: CAPTURE, RESTORE, FREEZE, and UNFREEZE.

---

## SET (Video Overlay Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## SET (Video Overlay Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

---

## SET (Video Overlay Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### **BRIGHTNESS level**

Sets the brightness to the specified level (0-100).

**M-Motion specific:** The default value is **80**.

### **CONTRAST level**

Sets the contrast to the specified level (0-100).

**M-Motion specific:** The default value is **90**.

### **GREYSCALE ON**

Enables greyscale. Video is displayed in black and white.

### **GREYSCALE OFF**

Disables greyscale. Video is displayed in color using the current settings.

**M-Motion specific:** The default is **off**.

### **HUE level**

Sets the hue to the specified level (0 - 100). A value of 50 specifies neutral hue, which is the default. Hue is also referred to as "tint."

**M-Motion specific:** The default is **50**.

### **IMAGE BITSPERPEL count**

Sets the number of bits per pixel for saving bit maps.

**M-Motion specific:** The default value is **12**.

### **IMAGE PELFORMAT type**

Sets the pel format or color encoding method for saving bit maps and images specified by the four-character code (FOURCC), such

as yuvb or rgbb.

**M-Motion specific:** The default is **yuvb**.

#### IMAGE FILE FORMAT format

Sets the specific image file format in which the image capture is to be stored (when "saved"). This format must be specified by a four-character code (FOURCC), for example, MMOT or OS13, and must be one of the currently supported and installed MMIO image file formats, or the device-specific format. This does not effect the loading or restoring of images. It overwrites any previous file-format value, such as that obtained through a LOAD operation.

**M-Motion specific:** The default is **MMOT**.

#### IMAGE COMPRESSION type

Sets the compression type to be used for saving images if possible. Possible values for **type** include: **none**.

**M-Motion specific:** The default is **none**.

The following **type** values are not supported:

- pic9
- pic16
- jpeg9
- jpegn
- rle4
- rle8

#### IMAGE QUALITY level

Sets the specified still image quality level. Used for optimizing the auto-selection of compression type when saving to file formats.

high	Indicates photo-like image quality.
med	Indicates that the image has moderate complexity or quality.
low	Indicates a lower color or complexity image.

**M-Motion specific:** The default is **high**.

#### SATURATION level

Sets the saturation to the specified level (0-100). Saturation is also referred to as "color."

**M-Motion specific:** The default value is **65**.

#### SHARPNESS level

Sets the sharpness to the specified level (0-100).

**M-Motion specific:** The default value is **80**.

#### VIDEO ON

Enables video output.

#### VIDEO OFF

Disables video output. The video window will be black. The following operations have no effect if video is set off: CAPTURE, RESTORE, FREEZE, and UNFREEZE.

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an **MM\_MCINOTIFY** message is sent to the application window procedure.

-----

## SET (Video Overlay Command) - Syntax Diagram

```
SET      object      BRIGHTNESS level
          CONTRAST level      WAIT
```

```
GREYSCALE ON
GREYSCALE OFF
HUE level
IMAGE BITSPERPEL count
IMAGE PELFORMAT type
SATURATION level
SHARPNESS level
IMAGE FILE FORMAT format
IMAGE COMPRESSION type
IMAGE QUALITY level
VIDEO ON
VIDEO OFF
```

NOTIFY

## Examples

# SET (Video Overlay Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

## STATUS

# STATUS (Video Overlay Command) - Example

```
status videooverlay hue wait
```

# STATUS (Video Overlay Command) - Purpose

The STATUS command obtains status information for the device and returns the current settings. These values might have been changed through previous SET operations, LOAD image operations, or the device defaults.

# STATUS (Video Overlay Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## STATUS (Video Overlay Command) Keyword - BRIGHTNESS

**BRIGHTNESS**

Returns the brightness level.

-----

## STATUS (Video Overlay Command) Keyword - CONTRAST

**CONTRAST**

Returns the contrast level.

-----

## STATUS (Video Overlay Command) Keyword - GREYSCALE

**GREYSCALE**

Returns **ON** or **OFF**.

-----

## STATUS (Video Overlay Command) Keyword - HORIZONTAL IMAGE EXTENT

**HORIZONTAL IMAGE EXTENT**

Returns the horizontal (X) source extent for the currently loaded image.

-----

## STATUS (Video Overlay Command) Keyword - HUE



## **HUE**

Returns the hue level.

---

# **STATUS (Video Overlay Command) Keyword - IMAGE BITSPERPEL**

## **IMAGE BITSPERPEL**

Returns the number of bits per pixel for saving bit maps and images.

---

# **STATUS (Video Overlay Command) Keyword - IMAGE COMPRESSION**

## **IMAGE COMPRESSION**

Returns the compression type used for saving bit maps and images.

---

# **STATUS (Video Overlay Command) Keyword - IMAGE FILE FORMAT**

## **IMAGE FILE FORMAT**

Returns the image file format.

---

# **STATUS (Video Overlay Command) Keyword - IMAGE PELFORMAT**

## **IMAGE PELFORMAT**

Returns the pel format for saving bit maps and images.

---

# **STATUS (Video Overlay Command) Keyword - IMAGE QUALITY**

**IMAGE QUALITY**  
Returns the still image quality level.

-----

## STATUS (Video Overlay Command) Keyword - MODE

**MODE**  
Returns **OTHER**.

-----

## STATUS (Video Overlay Command) Keyword - READY

**READY**  
Returns TRUE.

-----

## STATUS (Video Overlay Command) Keyword - SATURATION

**SATURATION**  
Returns the saturation level.

-----

## STATUS (Video Overlay Command) Keyword - SHARPNESS

**SHARPNESS**  
Returns the sharpness level.

-----

## STATUS (Video Overlay Command) Keyword - TIME FORMAT

**TIME FORMAT**  
Returns the time format.

-----

## STATUS (Video Overlay Command) Keyword -

# TRANSPARENT COLOR

## TRANSPARENT COLOR

Returns the transparency color value, relative to the transparency type. Video displays through all pels of this color.

**M-Motion specific:** Returns 3 (CLR\_PINK).

---

# STATUS (Video Overlay Command) Keyword - TRANSPARENT TYPE

## TRANSPARENT TYPE

Returns the type of transparency supported (if any), such as RGB value or palette entry.

**M-Motion specific:** Returns PALETTE.

---

# STATUS (Video Overlay Command) Keyword - VERTICAL IMAGE EXTENT

## VERTICAL IMAGE EXTENT

Returns the vertical (Y) source extent for the currently loaded image.

---

# STATUS (Video Overlay Command) Keyword - WINDOW HANDLE

## WINDOW HANDLE

Returns the window handle of the overlay video window in the low word of the return value.

---

# STATUS (Video Overlay Command) Keyword - WAIT

## WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

---

# STATUS (Video Overlay Command) Keyword - NOTIFY

## NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# STATUS (Video Overlay Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## BRIGHTNESS

Returns the brightness level.

## CONTRAST

Returns the contrast level.

## GREYSCALE

Returns **ON** or **OFF**.

## HORIZONTAL IMAGE EXTENT

Returns the horizontal (X) source extent for the currently loaded image.

## HUE

Returns the hue level.

## IMAGE BITSPERPEL

Returns the number of bits per pixel for saving bit maps and images.

## IMAGE COMPRESSION

Returns the compression type used for saving bit maps and images.

## IMAGE FILE FORMAT

Returns the image file format.

## IMAGE PELFORMAT

Returns the pel format for saving bit maps and images.

## IMAGE QUALITY

Returns the still image quality level.

## MODE

Returns **OTHER**.

## READY

Returns TRUE.

## SATURATION

Returns the saturation level.

## SHARPNESS

Returns the sharpness level.

## TIME FORMAT

Returns the time format.

## TRANSPARENT COLOR

Returns the transparency color value, relative to the transparency type. Video displays through all pels of this color.

**M-Motion specific:** Returns **3** (CLR\_PINK).

**TRANSPARENT TYPE**

Returns the type of transparency supported (if any), such as RGB value or palette entry.

**M-Motion specific:** Returns **PALETTE**.

**VERTICAL IMAGE EXTENT**

Returns the vertical (Y) source extent for the currently loaded image.

**WINDOW HANDLE**

Returns the window handle of the overlay video window in the low word of the return value.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

STATUS (Video Overlay Command) - Syntax Diagram



STATUS (Video Overlay Command) - Topics

- Select an item:
- Purpose
  - Syntax Diagram
  - Keywords
  - Example
  - Glossary

---

# UNFREEZE

---

## UNFREEZE (Video Overlay Command) - Example

```
unfreeze videooverlay at 100 100 260 220 wait
```

---

## UNFREEZE (Video Overlay Command) - Purpose

The UNFREEZE command stops updating the video buffer by the video source. Supported only if **can freeze** returns TRUE. (See the [CAPABILITY](#) command.)

---

## UNFREEZE (Video Overlay Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## UNFREEZE (Video Overlay Command) Keyword - AT rect

### AT rect

Specifies a rectangle relative to the window origin in device coordinates. The rectangle array is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner. Only the video in that subregion will be unfrozen.

---

## UNFREEZE (Video Overlay Command) Keyword - OUTSIDE rect

**OUTSIDE rect**

The area outside the specified rectangle is to be affected.

---

## UNFREEZE (Video Overlay Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## UNFREEZE (Video Overlay Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## UNFREEZE (Video Overlay Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**AT rect**

Specifies a rectangle relative to the window origin in device coordinates. The rectangle array is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner. Only the video in that subregion will be unfrozen.

**OUTSIDE rect**

The area outside the specified rectangle is to be affected.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## UNFREEZE (Video Overlay Command) - Syntax Diagram

UNFREEZE      object      AT rect  
OUTSIDE rect      WAIT  
NOTIFY

## Examples

# UNFREEZE (Video Overlay Command) - Topics

Select an item:

- [Purpose](#)
- [Syntax Diagram](#)
- [Keywords](#)
- [Example](#)
- [Glossary](#)

## WHERE

# WHERE (Video Overlay Command) - Example

```
where videooverlay source wait
```

# WHERE (Video Overlay Command) - Purpose

The WHERE command returns the source and destination rectangles set by the PUT command and returns the size and position of the video window.

# WHERE (Video Overlay Command) Keyword - object

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name



- Filename
- Alias

-----

## WHERE (Video Overlay Command) Keyword - SOURCE

### **SOURCE**

Returns the currently set source clipping rectangle, in the format X1 Y1 X2 Y2.

-----

## WHERE (Video Overlay Command) Keyword - DESTINATION

### **DESTINATION**

Returns a rectangle that describes where in the video window that video will be displayed in the format X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner.

-----

## WHERE (Video Overlay Command) Keyword - WINDOW

### **WINDOW**

Returns the current window position and size relative to the parent window in the format X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner.

-----

## WHERE (Video Overlay Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## WHERE (Video Overlay Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# WHERE (Video Overlay Command) - Keywords

- object**  
Object associated with this media control interface command. The object can be one of the following:
- Device type
  - Device name
  - Filename
  - Alias
- SOURCE**  
Returns the currently set source clipping rectangle, in the format X1 Y1 X2 Y2.
- DESTINATION**  
Returns a rectangle that describes where in the video window that video will be displayed in the format X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner.
- WINDOW**  
Returns the current window position and size relative to the parent window in the format X1 Y1 X2 Y2. The coordinates X1 Y1 specify the lower-left corner and X2 Y2 specify the upper-right corner.
- WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.
- NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# WHERE (Video Overlay Command) - Syntax Diagram

WHERE      [object](#)

[SOURCE](#)      [WAIT](#)  
[DESTINATION](#)  
[WINDOW](#)      [NOTIFY](#)



---

# WHERE (Video Overlay Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

# WINDOW

---

## WINDOW (Video Overlay Command) - Example

```
window videooverlay handle default wait
```

---

## WINDOW (Video Overlay Command) - Purpose

The WINDOW command controls the appearance of the video window.

---

## WINDOW (Video Overlay Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## WINDOW (Video Overlay Command) Keyword - HANDLE window\_handle

### **HANDLE window\_handle**

Sets a new window in which to play the video image. If the old window was the default window created by the driver, it is hidden. Otherwise, the application is responsible for managing the old window.

---

## WINDOW (Video Overlay Command) Keyword - HANDLE DEFAULT

#### **HANDLE DEFAULT**

Indicates the driver should display video in the window automatically created by the driver.

---

## **WINDOW (Video Overlay Command) Keyword - STATE ACTIVATE**

#### **STATE ACTIVATE**

Activates the default video window if it is a frame window. This has no other effect on other windows. The frame window is made the topmost window.

---

## **WINDOW (Video Overlay Command) Keyword - STATE DEACTIVATE**

#### **STATE DEACTIVATE**

Deactivates the default video window if it is a state window. This has no effect on other windows.

---

## **WINDOW (Video Overlay Command) Keyword - STATE HIDE**

#### **STATE HIDE**

Hides the default video window.

---

## **WINDOW (Video Overlay Command) Keyword - STATE MAXIMIZE**

#### **STATE MAXIMIZE**

Maximizes the default video window. This indicator has no effect if the window is in a maximized state, and is also mutually exclusive with STATE MINIMIZE and STATE RESTORE.

---

## **WINDOW (Video Overlay Command) Keyword - STATE MINIMIZE**

#### **STATE MINIMIZE**

Minimizes the default video window. This indicator has no effect if the window is in a minimized state, and is also mutually exclusive with STATE MAXIMIZE and STATE RESTORE.

---

## WINDOW (Video Overlay Command) Keyword - STATE RESTORE

### STATE RESTORE

Restores the default video window. This indicator has no effect if currently in a window and is mutually exclusive with STATE MAXIMIZE and STATE MINIMIZE.

---

## WINDOW (Video Overlay Command) Keyword - STATE SHOW

### STATE SHOW

Shows the default video window.

---

## WINDOW (Video Overlay Command) Keyword - TEXT caption

### TEXT caption

Specifies the text to display in the caption of the default video window.

---

## WINDOW (Video Overlay Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## WINDOW (Video Overlay Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# WINDOW (Video Overlay Command) - Keywords

**Note:** The STATE and TEXT keywords will not affect an application-supplied alternate window.

## **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## **HANDLE window\_handle**

Sets a new window in which to play the video image. If the old window was the default window created by the driver, it is hidden. Otherwise, the application is responsible for managing the old window.

## **HANDLE DEFAULT**

Indicates the driver should display video in the window automatically created by the driver.

## **STATE ACTIVATE**

Activates the default video window if it is a frame window. This has no other effect on other windows. The frame window is made the topmost window.

## **STATE DEACTIVATE**

Deactivates the default video window if it is a state window. This has no effect on other windows.

## **STATE HIDE**

Hides the default video window.

## **STATE MAXIMIZE**

Maximizes the default video window. This indicator has no effect if the window is in a maximized state, and is also mutually exclusive with STATE MINIMIZE and STATE RESTORE.

## **STATE MINIMIZE**

Minimizes the default video window. This indicator has no effect if the window is in a minimized state, and is also mutually exclusive with STATE MAXIMIZE and STATE RESTORE.

## **STATE RESTORE**

Restores the default video window. This indicator has no effect if currently in a window and is mutually exclusive with STATE MAXIMIZE and STATE MINIMIZE.

## **STATE SHOW**

Shows the default video window.

## **TEXT caption**

Specifies the text to display in the caption of the default video window.

## **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

## **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# WINDOW (Video Overlay Command) - Syntax Diagram

WINDOW	object	HANDLE window_handle	WAIT
		HANDLE DEFAULT	NOTIFY
		STATE ACTIVATE	
		STATE DEACTIVATE	
		STATE HIDE	
		STATE MAXIMIZE	
		STATE MINIMIZE	
		STATE RESTORE	
		STATE SHOW	
		TEXT caption	

Examples

-----

## WINDOW (Video Overlay Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

-----

## Waveform Audio Commands

The waveform audio device supports the following device-type specific commands and extensions to the following basic and required commands:

- [CAPABILITY](#)
- [CONNECTOR](#)
- [COPY](#)
- [CUE](#)
- [CUT](#)
- [DELETE](#)
- [LOAD](#)
- [PASTE](#)
- [RECORD](#)
- [REDO](#)
- [SEEK](#)
- [SET](#)
- [STATUS](#)
- [UNDO](#)

-----

## CAPABILITY

-----

## CAPABILITY (Waveaudio Command) - Example

```
capability waveaudio extended format bitspersample 16
samplespersec 11025 tag PCM channels 2 mode play
```

---

## CAPABILITY (Waveaudio Command) - Purpose

The CAPABILITY command requests additional information about the capabilities of waveform audio device driver.

---

## CAPABILITY (Waveaudio Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

## CAPABILITY (Waveaudio Command) Keyword - CAN EJECT

### **CAN EJECT**

Returns FALSE. Wave audio devices cannot eject the media.

---

## CAPABILITY (Waveaudio Command) Keyword - CAN PLAY

### **CAN PLAY**

Returns TRUE.

---

## CAPABILITY (Waveaudio Command) Keyword - CAN RECORD



**CAN RECORD**

Returns TRUE if the current audio device supports recording.

---

## CAPABILITY (Waveaudio Command) Keyword - CAN SAVE

**CAN SAVE**

Returns TRUE if the wave audio device can save data.

---

## CAPABILITY (Waveaudio Command) Keyword - CAN SETVOLUME

**CAN SETVOLUME**

Returns TRUE if the device supports software control of volume level.

---

## CAPABILITY (Waveaudio Command) Keyword - COMPOUND DEVICE

**COMPOUND DEVICE**

Returns TRUE. Wave audio devices are compound devices.

---

## CAPABILITY (Waveaudio Command) Keyword - DEVICE TYPE

**DEVICE TYPE**

Returns **waveaudio**.

---

## CAPABILITY (Waveaudio Command) Keyword - EXTENDED

**EXTENDED**

Indicates that extended capabilities are to be queried.

---

# CAPABILITY (Waveaudio Command) Keyword - FORMAT

**FORMAT**  
Indicates that waveaudio format will be queried. If FORMAT is specified, BITSPERSAMPLE, SAMPLESPERSEC, TAG, and CHANNELS must also be specified.

---

## CAPABILITY (Waveaudio Command) Keyword - BITSPERSAMPLE integer

**BITSPERSAMPLE integer**  
The integer indicates the number of bits in a waveaudio sample (typically 8 or 16).

---

## CAPABILITY (Waveaudio Command) Keyword - SAMPLESPERSEC integer

**SAMPLESPERSEC integer**  
The integer indicates the number of samples per second the waveaudio will utilize.

---

## CAPABILITY (Waveaudio Command) Keyword - TAG type

**TAG type**  
The type is a valid format tag which can be used with set (see the "[SET](#) object FORMAT TAG" command).

---

## CAPABILITY (Waveaudio Command) Keyword - CHANNELS integer

**CHANNELS integer**  
Where integer indicates the number of channels (typically 1 or 2).

---

## CAPABILITY (Waveaudio Command) Keyword - MODE type

**MODE type**

Where type is either **play** or **record**.

---

## CAPABILITY (Waveaudio Command) Keyword - HAS AUDIO

**HAS AUDIO**

Returns TRUE.

---

## CAPABILITY (Waveaudio Command) Keyword - HAS VIDEO

**HAS VIDEO**

Returns FALSE. Wave audio devices do not support video.

---

## CAPABILITY (Waveaudio Command) Keyword - MESSAGE command

**MESSAGE command**

Returns TRUE if the device supports the command specified by **command**. The **command** can be any string command such as OPEN, PLAY, and so on.

---

## CAPABILITY (Waveaudio Command) Keyword - PREROLL TIME

**PREROLL TIME**

Returns **0**, indicating the preroll time is not bounded.

---

## CAPABILITY (Waveaudio Command) Keyword - PREROLL TYPE

**PREROLL TYPE**

Returns the preroll characteristics of the device: Returns **notified**.

---

## CAPABILITY (Waveaudio Command) Keyword - USES FILES

### USES FILES

Returns TRUE. Wave audio devices use files for operation.

---

## CAPABILITY (Waveaudio Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

---

## CAPABILITY (Waveaudio Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CAPABILITY (Waveaudio Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### CAN EJECT

Returns FALSE. Wave audio devices cannot eject the media.

### CAN PLAY

Returns TRUE.

### CAN RECORD

Returns TRUE if the current audio device supports recording.

### CAN SAVE

Returns TRUE if the wave audio device can save data.

### CAN SETVOLUME

Returns TRUE if the device supports software control of volume level.

#### COMPOUND DEVICE

Returns TRUE. Wave audio devices are compound devices.

#### DEVICE TYPE

Returns **waveaudio**.

#### EXTENDED

Indicates that extended capabilities are to be queried.

#### FORMAT

Indicates that waveaudio format will be queried. If FORMAT is specified, BITSPERSAMPLE, SAMPLESPERSEC, TAG, and CHANNELS must also be specified.

##### BITSPERSAMPLE integer

The integer indicates the number of bits in a waveaudio sample (typically 8 or 16).

##### SAMPLESPERSEC integer

The integer indicates the number of samples per second the waveaudio will utilize.

##### TAG type

The type is a valid format tag which can be used with set (see the "[SET](#) object FORMAT TAG" command).

##### CHANNELS integer

Where integer indicates the number of channels (typically 1 or 2).

##### MODE type

Where type is either **play** or **record**.

#### HAS AUDIO

Returns TRUE.

#### HAS VIDEO

Returns FALSE. Wave audio devices do not support video.

#### MESSAGE command

Returns TRUE if the device supports the command specified by **command**. The **command** can be any string command such as OPEN, PLAY, and so on.

#### PREROLL TIME

Returns **0**, indicating the preroll time is not bounded.

#### PREROLL TYPE

Returns the preroll characteristics of the device: Returns **notified**.

#### USES FILES

Returns TRUE. Wave audio devices use files for operation.

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CAPABILITY (Waveaudio Command) - Syntax Diagram

CAPABILITY      [object](#)

[CAN EJECT](#)  
[CAN PLAY](#)  
[CAN RECORD](#)  
[CAN SAVE](#)

[WAIT](#)  
[NOTIFY](#)

```
CAN SETVOLUME
COMPOUND DEVICE
DEVICE TYPE

EXTENDED      FORMAT      BITSPERSAMPLE integer
                                SAMPLESPERSEC integer
                                TAG type
                                CHANNELS integer
                                MODE type

HAS AUDIO
HAS VIDEO
MESSAGE command
PREROLL TIME
PREROLL TYPE
USES FILES
```

Examples

-----

## CAPABILITY (Waveaudio Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

-----

## CONNECTOR

-----

## CONNECTOR (Waveaudio Command) - Example

```
connector waveaudio enable type line in wait
```

-----

## CONNECTOR (Waveaudio Command) - Purpose

The CONNECTOR command enables, disables, or queries the status of connector on a device.

-----

## CONNECTOR (Waveaudio Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## CONNECTOR (Waveaudio Command) Keyword - ENABLE

**ENABLE**

Enables information flow through the indicated connector. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

-----

## CONNECTOR (Waveaudio Command) Keyword - DISABLE

**DISABLE**

Disables information flow through the indicated connector. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

-----

## CONNECTOR (Waveaudio Command) Keyword - QUERY

**QUERY**

Queries the state of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

-----

## CONNECTOR (Waveaudio Command) Keyword - NUMBER connector\_number

**NUMBER connector\_number**

Indicates the connector number on which to perform the requested action. If this item is omitted, then the first connector is assumed. If the TYPE item is included, then the connector number is interpreted as a relative offset within the specified connector type.

-----

## CONNECTOR (Waveaudio Command) Keyword - TYPE

# connector\_type

## TYPE connector\_type

Indicates the type of connector to which the requested action applies. The following connector types are directly supported by this device.

wave stream

Digital input or output for the audio amplifier/mixer. This connector is always enabled.

The waveform audio device also recognizes the following connector types and will attempt to control the corresponding amp/mixer connector if the amp/mixer provides the support.

line in

The line input connector. This connector is usually attached to the line out connector of another device such as a tape player or other audio input source.

microphone

The microphone connector. This connector is usually attached to a microphone for live recording or voice annotation.

line out

The line output connector. This connector is usually attached to the line in connector of another device such as a tape recorder or other audio device.

speakers

The speakers connector. This connector is usually attached to a pair of external or internal speakers.

headphones

The headphones connector. This connector is usually attached to a pair of headphones.

-----

## CONNECTOR (Waveaudio Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## CONNECTOR (Waveaudio Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CONNECTOR (Waveaudio Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type



- Device name
- Filename
- Alias

#### ENABLE

Enables information flow through the indicated connector. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

#### DISABLE

Disables information flow through the indicated connector. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

#### QUERY

Queries the state of the indicated connector. The return value will be either TRUE or FALSE to indicate enabled or disabled respectively. Use of this keyword requires that the NUMBER or TYPE keywords must also be specified.

#### NUMBER connector\_number

Indicates the connector number on which to perform the requested action. If this item is omitted, then the first connector is assumed. If the TYPE item is included, then the connector number is interpreted as a relative offset within the specified connector type.

#### TYPE connector\_type

Indicates the type of connector to which the requested action applies. The following connector types are directly supported by this device.

wave stream

Digital input or output for the audio amplifier/mixer. This connector is always enabled.

The waveform audio device also recognizes the following connector types and will attempt to control the corresponding amp/mixer connector if the amp/mixer provides the support.

line in

The line input connector. This connector is usually attached to the line out connector of another device such as a tape player or other audio input source.

microphone

The microphone connector. This connector is usually attached to a microphone for live recording or voice annotation.

line out

The line output connector. This connector is usually attached to the line in connector of another device such as a tape recorder or other audio device.

speakers

The speakers connector. This connector is usually attached to a pair of external or internal speakers.

headphones

The headphones connector. This connector is usually attached to a pair of headphones.

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## CONNECTOR (Waveaudio Command) - Syntax Diagram

```
CONNECTOR      object      ENABLE
                  DISABLE
                  QUERY

NUMBER connector_number
```

`TYPE connector_type`

`WAIT  
NOTIFY`

## Examples

---

# CONNECTOR (Waveaudio Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## COPY

---

# COPY (Waveaudio Command) - Example

```
copy waveaudio from 1000 to 5000
```

---

# COPY (Waveaudio Command) - Purpose

The COPY command copies information from a file into the clipboard.

---

# COPY (Waveaudio Command) Keyword - object

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

## COPY (Waveaudio Command) Keyword - FROM pos

### FROM pos

The position to start copying. If FROM is omitted, the copy starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

---

## COPY (Waveaudio Command) Keyword - TO pos

### TO pos

The position to stop copying. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

---

## COPY (Waveaudio Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## COPY (Waveaudio Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## COPY (Waveaudio Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### FROM pos

The position to start copying. If FROM is omitted, the copy starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

**TO pos**  
The position to stop copying. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

**WAIT**  
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**  
The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## COPY (Waveaudio Command) - Syntax Diagram

COPY      [object](#)      [FROM pos](#)      [WAIT](#)  
                                 [TO pos](#)      [NOTIFY](#)



---

## COPY (Waveaudio Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

## CUE

---

## CUE (Waveaudio Command) - Example

`cue waveaudio input wait`

---

## CUE (Waveaudio Command) - Purpose

The CUE command prepares for playback or recording. The CUE command does not have to be issued prior to playback or recording. However, depending on the device, it can reduce the delay associated with the PLAY or RECORD command.

The CUE command is not related to the [SETCUEPOINT](#) command.

---

## CUE (Waveaudio Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CUE (Waveaudio Command) Keyword - INPUT

### **INPUT**

Prepares the input for recording.

---

## CUE (Waveaudio Command) Keyword - OUTPUT

### **OUTPUT**

Prepares the output for playback. This is the default.

---

## CUE (Waveaudio Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## CUE (Waveaudio Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# CUE (Waveaudio Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**INPUT**

Prepares the input for recording.

**OUTPUT**

Prepares the output for playback. This is the default.

**WAIT**

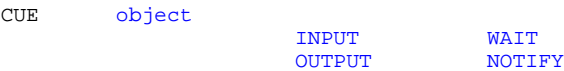
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# CUE (Waveaudio Command) - Syntax Diagram



-----

# CUE (Waveaudio Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)

---

# CUT

---

## CUT (Waveaudio Command) - Example

```
cut waveaudio from 1000 to 4000 wait
```

---

## CUT (Waveaudio Command) - Purpose

The CUT command cuts removes the specified range and places the data in the clipboard.

---

## CUT (Waveaudio Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## CUT (Waveaudio Command) Keyword - FROM pos

### **FROM pos**

The position to start cutting. If FROM is omitted, the cut starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

---

## CUT (Waveaudio Command) Keyword - TO pos

**TO pos**

The position to stop cutting. If FROM is omitted, the cut starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

---

## CUT (Waveaudio Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## CUT (Waveaudio Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CUT (Waveaudio Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**FROM pos**

The position to start cutting. If FROM is omitted, the cut starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

**TO pos**

The position to stop cutting. If FROM is omitted, the cut starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## CUT (Waveaudio Command) - Syntax Diagram



CUT      [object](#)

[FROM pos](#)      [WAIT](#)  
[TO pos](#)      [NOTIFY](#)

## Examples

---

## CUT (Waveaudio Command) - Topics

Select an item:

- [Purpose](#)
- [Syntax Diagram](#)
- [Keywords](#)
- [Example](#)
- [Glossary](#)

---

## DELETE

---

## DELETE (Waveaudio Command) - Example

```
delete waveaudio from 1000 to 4000 wait
```

---

## DELETE (Waveaudio Command) - Purpose

The DELETE command deletes information from a file.

---

## DELETE (Waveaudio Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type

- Device name
- Filename
- Alias

-----

## DELETE (Waveaudio Command) Keyword - FROM pos

### **FROM pos**

The position to start deleting. If FROM is omitted, the delete starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

-----

## DELETE (Waveaudio Command) Keyword - TO pos

### **TO pos**

The position to stop deleting. If FROM is omitted, the delete starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

-----

## DELETE (Waveaudio Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## DELETE (Waveaudio Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## DELETE (Waveaudio Command) - Keywords

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type

- Device name
- Filename
- Alias

#### FROM pos

The position to start deleting. If FROM is omitted, the delete starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

#### TO pos

The position to stop deleting. If FROM is omitted, the delete starts from the current position. If TO is omitted, the end of file is assumed. The position of the media will either be the from position if FROM is specified, or the previous position if FROM is not specified.

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## DELETE (Waveaudio Command) - Syntax Diagram

DELETE      [object](#)

[FROM pos](#)                      [WAIT](#)

[TO pos](#)                         [NOTIFY](#)

Examples

-----

## DELETE (Waveaudio Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

-----

## LOAD

-----

# LOAD (Waveaudio Command) - Example

```
load waveaudio bells.wav wait
```

---

## LOAD (Waveaudio Command) - Purpose

The LOAD command loads a new device element (file) into an already open device context.

---

## LOAD (Waveaudio Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

---

## LOAD (Waveaudio Command) Keyword - filename

### **filename**

The name of the file to load. Optional if OPEN is specified.

---

## LOAD (Waveaudio Command) Keyword - READONLY

### **READONLY**

The system will open the file in a read-only mode to prevent any inadvertent modifications to the file. The waveaudio driver might also be able to improve load and run time performance as no modifications will be allowed. This flag can only be specified in conjunction with a file element. Specifying the READONLY keyword will disable support for the SAVE, RECORD, CUT, DELETE, and PASTE commands.

---

## LOAD (Waveaudio Command) Keyword - NEW

#### NEW

A temporary element is created for subsequent use with [MCI\\_RECORD](#), [MCI\\_PLAY](#), and other commands. The temporary file can be made permanent by providing a name using the [MCI\\_SAVE](#) message.

---

## LOAD (Waveaudio Command) Keyword - WAIT

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## LOAD (Waveaudio Command) Keyword - NOTIFY

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## LOAD (Waveaudio Command) - Keywords

#### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

#### filename

The name of the file to load. Optional if OPEN is specified.

#### READONLY

The system will open the file in a read-only mode to prevent any inadvertent modifications to the file. The waveaudio driver might also be able to improve load and run time performance as no modifications will be allowed. This flag can only be specified in conjunction with a file element. Specifying the READONLY keyword will disable support for the SAVE, RECORD, CUT, DELETE, and PASTE commands.

#### NEW

A temporary element is created for subsequent use with [MCI\\_RECORD](#), [MCI\\_PLAY](#), and other commands. The temporary file can be made permanent by providing a name using the [MCI\\_SAVE](#) message.

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## LOAD (Waveaudio Command) - Syntax Diagram

LOAD      **object**      **filename**      READONLY  
NEW  
WAIT  
NOTIFY

Examples

---

## LOAD (Waveaudio Command) - Topics

Select an item:  
[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## PASTE

---

## PASTE (Waveaudio Command) - Example

```
paste waveaudio from 1000 to 4000 wait
```

---

## PASTE (Waveaudio Command) - Purpose

The PASTE command pastes information from the clipboard into a file. The media position after a paste operation is at the end of the pasted data.

---

## PASTE (Waveaudio Command) Keyword - object

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## PASTE (Waveaudio Command) Keyword - FROM pos

**FROM pos**

The position to start pasting. If FROM is omitted, the paste starts at the current position.

-----

## PASTE (Waveaudio Command) Keyword - TO pos

**TO pos**

The position to stop pasting. The pasted data *replaces* data from the FROM position (or the current position if FROM is not specified) to the TO position.

If TO is omitted, the end of file is assumed and the pasted data is *inserted* starting at the FROM position (or the current position if FROM is not specified).

-----

## PASTE (Waveaudio Command) Keyword - CONVERT

**CONVERT**

Converts data in the clipboard to the current file format.

-----

## PASTE (Waveaudio Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## PASTE (Waveaudio Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# PASTE (Waveaudio Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**FROM pos**

The position to start pasting. If FROM is omitted, the paste starts at the current position.

**TO pos**

The position to stop pasting. The pasted data *replaces* data from the FROM position (or the current position if FROM is not specified) to the TO position.

If TO is omitted, the end of file is assumed and the pasted data is *inserted* starting at the FROM position (or the current position if FROM is not specified).

**CONVERT**

Converts data in the clipboard to the current file format.

**WAIT**

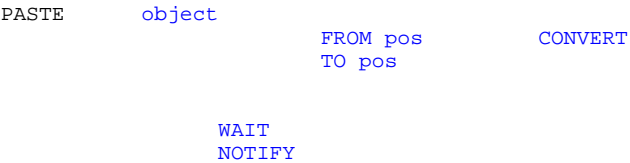
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

# PASTE (Waveaudio Command) - Syntax Diagram



-----

# PASTE (Waveaudio Command) - Topics



Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)  
[Example](#)  
[Glossary](#)

---

## RECORD

---

### RECORD (Waveaudio Command) - Example

```
record waveaudio to 1000 overwrite wait
```

---

### RECORD (Waveaudio Command) - Purpose

The RECORD command start recording audio. Recording does not overwrite existing data. New data is inserted at the current position. All data recorded after a file is opened is discarded if the file is closed without saving the data.

---

### RECORD (Waveaudio Command) Keyword - object

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

### RECORD (Waveaudio Command) Keyword - FROM pos

#### **FROM pos**

The position to start recording. If FROM is omitted, the device starts recording at the current position; if TO is omitted, the device records until a STOP or PAUSE command is received.

---

## RECORD (Waveaudio Command) Keyword - TO pos

### TO pos

The position to stop recording. If FROM is omitted, the device starts recording at the current position; if TO is omitted, the device records until a STOP or PAUSE command is received.

---

## RECORD (Waveaudio Command) Keyword - INSERT

### INSERT

New data is added to the device element.

---

## RECORD (Waveaudio Command) Keyword - OVERWRITE

### OVERWRITE

New data will replace data in the device element.

---

## RECORD (Waveaudio Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## RECORD (Waveaudio Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## RECORD (Waveaudio Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**FROM pos**

The position to start recording. If FROM is omitted, the device starts recording at the current position; if TO is omitted, the device records until a STOP or PAUSE command is received.

**TO pos**

The position to stop recording. If FROM is omitted, the device starts recording at the current position; if TO is omitted, the device records until a STOP or PAUSE command is received.

**INSERT**

New data is added to the device element.

**OVERWRITE**

New data will replace data in the device element.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## RECORD (Waveaudio Command) - Syntax Diagram

```
RECORD      object

            FROM pos
            TO pos

            INSERT
            OVERWRITE

            WAIT
            NOTIFY
```

Examples

-----

## RECORD (Waveaudio Command) - Remarks

This command requires that a device element be loaded prior to recording. See the [LOAD](#) command for more information.

-----

## RECORD (Waveaudio Command) - Topics

Select an item:

[Purpose](#)

[Syntax Diagram](#)

[Keywords](#)

[Remarks](#)

[Example](#)

[Glossary](#)

---

## REDO

---

### REDO (Waveaudio Command) - Example

```
redo waveaudio wait
```

---

### REDO (Waveaudio Command) - Purpose

The REDO command redoes the last editing action (cut, paste, or delete) which was undone with the [UNDO](#) command. REDO should immediately follow [UNDO](#); otherwise, editing actions performed after [UNDO](#) (and before a corresponding REDO) will be lost when the REDO command is issued. The position of the media after a redo operation is 0.

Multiple REDO operations are permitted, corresponding to the number of editing operations that have been previously undone with the [UNDO](#) command.

---

### REDO (Waveaudio Command) Keyword - object

#### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

### REDO (Waveaudio Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## REDO (Waveaudio Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## REDO (Waveaudio Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

-----

## REDO (Waveaudio Command) - Syntax Diagram

REDO      [object](#)      [WAIT](#)  
                                 [NOTIFY](#)

**Examples**

-----

## REDO (Waveaudio Command) - Topics

Select an item:

[Purpose](#)  
[Syntax Diagram](#)  
[Keywords](#)

---

# SEEK

---

## SEEK (Waveaudio Command) - Example

```
seek waveaudio to start wait
```

---

## SEEK (Waveaudio Command) - Purpose

The SEEK command seeks finds the specified location in the file.

---

## SEEK (Waveaudio Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## SEEK (Waveaudio Command) Keyword - TO pos

### **TO pos**

Specifies the final position for the seek.

---

## SEEK (Waveaudio Command) Keyword - TO START

**TO START**

Seeks to the start of the file.

-----

## SEEK (Waveaudio Command) Keyword - TO END

**TO END**

Seeks to the end of the file.

-----

## SEEK (Waveaudio Command) Keyword - WAIT

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## SEEK (Waveaudio Command) Keyword - NOTIFY

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

-----

## SEEK (Waveaudio Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**TO pos**

Specifies the final position for the seek.

**TO START**

Seeks to the start of the file.

**TO END**

Seeks to the end of the file.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an MM\_MCINOTIFY message is sent to the application window procedure.

---

## SEEK (Waveaudio Command) - Syntax Diagram

```
SEEK      object      TO pos
                        TO START
                        TO END      WAIT
                                    NOTIFY
```



---

## SEEK (Waveaudio Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

## SET

---

## SET (Waveaudio Command) - Example

```
set waveaudio channels 2 wait
```

---

## SET (Waveaudio Command) - Purpose

The SET command sets the various control and attribute items.

---

## SET (Waveaudio Command) Keyword - object



**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

-----

## SET (Waveaudio Command) Keyword - AUDIO

**AUDIO**

Specifies the audio attributes of the device context determined by the ALL, LEFT, RIGHT, OVER and VOLUME keywords.

-----

## SET (Waveaudio Command) Keyword - ALL

**ALL**

Applies to both or all of the channels (default).

Specify ON or OFF with the ALL keyword.

-----

## SET (Waveaudio Command) Keyword - ON

**ON**

Enables audio.

-----

## SET (Waveaudio Command) Keyword - OFF

**OFF**

Disables audio.

-----

## SET (Waveaudio Command) Keyword - LEFT

**LEFT**

Applies to the left channel.

Specify ON or OFF with the LEFT keyword.

-----

## SET (Waveaudio Command) Keyword - ON

**ON**

Enables audio to the left channel.

-----

## SET (Waveaudio Command) Keyword - OFF

**OFF**

Disables audio to the left channel.

-----

## SET (Waveaudio Command) Keyword - RIGHT

**RIGHT**

Applies to the right channel.

Specify ON or OFF with the RIGHT keyword.

-----

## SET (Waveaudio Command) Keyword - ON

**ON**

Enables audio to the right channel.

-----

## SET (Waveaudio Command) Keyword - OFF

**OFF**

Disables audio to the right channel.

-----

## SET (Waveaudio Command) Keyword - OVER milliseconds

**OVER milliseconds**

Applies the change over the specified time period (fade).

---

## SET (Waveaudio Command) Keyword - VOLUME percentage

**VOLUME percentage**

Sets the volume level.

---

## SET (Waveaudio Command) Keyword - BITSPERSAMPLE integer

**BITSPERSAMPLE integer**

The number of bits per sample to be played or recorded. The file is saved in this format.

---

## SET (Waveaudio Command) Keyword - BYTESPERSEC integer

**BYTESPERSEC integer**

The average number of bytes per second to be played or recorded. The file is saved in this format.

---

## SET (Waveaudio Command) Keyword - CHANNELS integer

**CHANNELS integer**

The channel count for playing and recording. The file is saved in this format.

---

## SET (Waveaudio Command) Keyword - FORMAT TAG tag

**FORMAT TAG tag**

The format type for playing and recording. The file is saved in this format. If the waveform format is being changed then the SET command should be sent first with the **FORMAT TAG** keyword specified as the driver might need to change the other settings to be compatible with the new waveform format. After setting the waveform format, the other parameters can be set as necessary within

the currently selected waveform format. An error will be returned if the requested change results in an unsupported configuration.

An application can use the [STATUS](#) message to see if any of the other settings were changed to maintain a valid configuration. The following **tag** values are defined:

pcm	The format type of PCM (pulse code modulation) for playing and recording. The file is saved in this format.
avc adpcm	The IBM AVC ADPCM (adaptive differential pulse code modulation) format type for playing and recording. The file is saved in this format.
microsoft adpcm	The Microsoft ADPCM format type for playing and recording. The file is saved in this format.
cvsd	The IBM Speech Viewer format type for playing and recording. The file is saved in this format.
alaw	The CCITT A-Law format type for playing and recording. The file is saved in this format.
mulaw	The CCITT MuLaw format type for playing and recording. The file is saved in this format.
ibm alaw	The IBM A-Law format type for playing and recording. This format type is the same as CCITT A-Law.
ibm mulaw	The IBM A-Law format type for playing and recording. This format type is the same as CCITT Mulaw.
ibm adpcm	The IBM ADPCM format type for playing and recording. The file is saved in this format.
oki adpcm	The OKI ADPCM format type for playing and recording. The file is saved in this format.
dvi adpcm	The DVI ADPCM format type for playing and recording. The file is saved in this format.
ct adpcm	The format type of Creative Labs ADPCM for playing and recording. The file is saved in this format.
digistd	The IBM Digispeech standard format type for playing and recording. The file is saved in this format.
digifix	The IBM Digispeech fixed format type for playing and recording. The file is saved in this format.

-----

## SET (Waveaudio Command) Keyword - SAMPLESPERSEC integer

### **SAMPLESPERSEC integer**

The sample rate for playing and recording. The file is saved in this format.

-----

## SET (Waveaudio Command) Keyword - TIME FORMAT BYTES

#### TIME FORMAT BYTES

Sets the time format to bytes. All position information is specified as bytes following this command.

-----

## SET (Waveaudio Command) Keyword - TIME FORMAT MILLISECONDS

#### TIME FORMAT MILLISECONDS

Sets the time format to milliseconds. All position information is specified as milliseconds following this command. You can abbreviate milliseconds as **ms**.

-----

## SET (Waveaudio Command) Keyword - TIME FORMAT MMTIME

#### TIME FORMAT MMTIME

Set the time format to MMTIME.

-----

## SET (Waveaudio Command) Keyword - TIME FORMAT SAMPLES

#### TIME FORMAT SAMPLES

Sets the time format to samples. All position information is specified as samples following this command.

-----

## SET (Waveaudio Command) Keyword - WAIT

#### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

-----

## SET (Waveaudio Command) Keyword - NOTIFY

#### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# SET (Waveaudio Command) - Keywords

## object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

## AUDIO

Specifies the audio attributes of the device context determined by the ALL, LEFT, RIGHT, OVER and VOLUME keywords.

### ALL

Applies to both or all of the channels (default).

Specify ON or OFF with the ALL keyword.

#### ON

Enables audio.

#### OFF

Disables audio.

### LEFT

Applies to the left channel.

Specify ON or OFF with the LEFT keyword.

#### ON

Enables audio to the left channel.

#### OFF

Disables audio to the left channel.

### RIGHT

Applies to the right channel.

Specify ON or OFF with the RIGHT keyword.

#### ON

Enables audio to the right channel.

#### OFF

Disables audio to the right channel.

### OVER milliseconds

Applies the change over the specified time period (fade).

### VOLUME percentage

Sets the volume level.

## BITSPERSAMPLE integer

The number of bits per sample to be played or recorded. The file is saved in this format.

## BYTESPERSEC integer

The average number of bytes per second to be played or recorded. The file is saved in this format.

## CHANNELS integer

The channel count for playing and recording. The file is saved in this format.

## FORMAT TAG tag

The format type for playing and recording. The file is saved in this format. If the waveform format is being changed then the SET command should be sent first with the **FORMAT TAG** keyword specified as the driver might need to change the other settings to be

compatible with the new waveform format. After setting the waveform format, the other parameters can be set as necessary within the currently selected waveform format. An error will be returned if the requested change results in an unsupported configuration.

An application can use the [STATUS](#) message to see if any of the other settings were changed to maintain a valid configuration. The following **tag** values are defined:

pcm	The format type of PCM (pulse code modulation) for playing and recording. The file is saved in this format.
avc adpcm	The IBM AVC ADPCM (adaptive differential pulse code modulation) format type for playing and recording. The file is saved in this format.
microsoft adpcm	The Microsoft ADPCM format type for playing and recording. The file is saved in this format.
cvsd	The IBM Speech Viewer format type for playing and recording. The file is saved in this format.
alaw	The CCITT A-Law format type for playing and recording. The file is saved in this format.
mulaw	The CCITT MuLaw format type for playing and recording. The file is saved in this format.
ibm alaw	The IBM A-Law format type for playing and recording. This format type is the same as CCITT A-Law.
ibm mulaw	The IBM A-Law format type for playing and recording. This format type is the same as CCITT Mulaw.
ibm adpcm	The IBM ADPCM format type for playing and recording. The file is saved in this format.
oki adpcm	The OKI ADPCM format type for playing and recording. The file is saved in this format.
dvi adpcm	The DVI ADPCM format type for playing and recording. The file is saved in this format.
ct adpcm	The format type of Creative Labs ADPCM for playing and recording. The file is saved in this format.
digistd	The IBM Digispeech standard format type for playing and recording. The file is saved in this format.
digifix	The IBM Digispeech fixed format type for playing and recording. The file is saved in this format.

#### **SAMPLESPERSEC integer**

The sample rate for playing and recording. The file is saved in this format.

#### **TIME FORMAT BYTES**

Sets the time format to bytes. All position information is specified as bytes following this command.

#### **TIME FORMAT MILLISECONDS**

Sets the time format to milliseconds. All position information is specified as milliseconds following this command. You can abbreviate milliseconds as **ms**.

#### **TIME FORMAT MMTIME**

Set the time format to MMTIME.

#### **TIME FORMAT SAMPLES**

Sets the time format to samples. All position information is specified as samples following this command.

#### **WAIT**

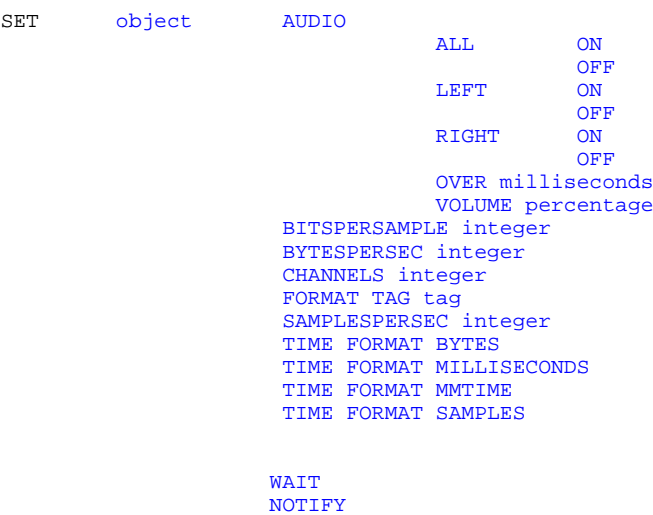
The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

#### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

# SET (Waveaudio Command) - Syntax Diagram



Examples

---

# SET (Waveaudio Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

# STATUS

---

# STATUS (Waveaudio Command) - Example

status waveaudio mode wait



---

## STATUS (Waveaudio Command) - Purpose

The STATUS command obtains status information for the device.

---

## STATUS (Waveaudio Command) Keyword - object

### **object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## STATUS (Waveaudio Command) Keyword - ALIGNMENT

### **ALIGNMENT**

Returns the block alignment of data in bytes.

---

## STATUS (Waveaudio Command) Keyword - BITSPERSAMPLE

### **BITSPERSAMPLE**

Returns the bits per sample.

---

## STATUS (Waveaudio Command) Keyword - BYTESPERSEC

### **BYTESPERSEC**

Returns the average number of bytes per second played or recorded.

---

## STATUS (Waveaudio Command) Keyword - CHANNELS

**CHANNELS**

Returns the number of channels set (1 for monaural, 2 for stereo).

-----

**STATUS (Waveaudio Command) Keyword - CURRENT TRACK**

**CURRENT TRACK**

Returns 1 for the current track.

-----

**STATUS (Waveaudio Command) Keyword - FORMAT TAG**

**FORMAT TAG**

Returns the format tag.

-----

**STATUS (Waveaudio Command) Keyword - LENGTH**

**LENGTH**

Returns the total length of the waveform.

-----

**STATUS (Waveaudio Command) Keyword - LENGTH TRACK track\_number**

**LENGTH TRACK track\_number**

Returns the length of the specified track.

-----

**STATUS (Waveaudio Command) Keyword - LEVEL**

**LEVEL**

Returns the current audio sample value.

-----

**STATUS (Waveaudio Command) Keyword - MEDIA**

# PRESENT

## MEDIA PRESENT

Returns MCI\_TRUE.

---

# STATUS (Waveaudio Command) Keyword - MODE

## MODE

Returns the current mode of the device: **not ready**, **stopped**, **playing**, **seeking**, **recording**, **paused**, or **other**.

---

# STATUS (Waveaudio Command) Keyword - NUMBER OF TRACKS

## NUMBER OF TRACKS

Returns the number of tracks.

---

# STATUS (Waveaudio Command) Keyword - POSITION

## POSITION

Returns the current position.

---

# STATUS (Waveaudio Command) Keyword - POSITION TRACK track\_number

## POSITION TRACK track\_number

Returns the position of the track specified by **track\_number**.

---

# STATUS (Waveaudio Command) Keyword - READY

## READY

Returns MCI\_TRUE if the device is ready.

---

## STATUS (Waveaudio Command) Keyword - SAMPLESPERSEC

### **SAMPLESPERSEC**

Returns the number of samples per second played or recorded.

---

## STATUS (Waveaudio Command) Keyword - TIME FORMAT

### **TIME FORMAT**

Returns the time format.

---

## STATUS (Waveaudio Command) Keyword - VOLUME

### **VOLUME**

Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

---

## STATUS (Waveaudio Command) Keyword - WAIT

### **WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

---

## STATUS (Waveaudio Command) Keyword - NOTIFY

### **NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## STATUS (Waveaudio Command) - Keywords

**object**

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

**ALIGNMENT**

Returns the block alignment of data in bytes.

**BITSPERSAMPLE**

Returns the bits per sample.

**BYTESPERSEC**

Returns the average number of bytes per second played or recorded.

**CHANNELS**

Returns the number of channels set (1 for monaural, 2 for stereo).

**CURRENT TRACK**

Returns 1 for the current track.

**FORMAT TAG**

Returns the format tag.

**LENGTH**

Returns the total length of the waveform.

**LENGTH TRACK *track\_number***

Returns the length of the specified track.

**LEVEL**

Returns the current audio sample value.

**MEDIA PRESENT**

Returns MCI\_TRUE.

**MODE**

Returns the current mode of the device: **not ready**, **stopped**, **playing**, **seeking**, **recording**, **paused**, or **other**.

**NUMBER OF TRACKS**

Returns the number of tracks.

**POSITION**

Returns the current position.

**POSITION TRACK *track\_number***

Returns the position of the track specified by **track\_number**.

**READY**

Returns MCI\_TRUE if the device is ready.

**SAMPLESPERSEC**

Returns the number of samples per second played or recorded.

**TIME FORMAT**

Returns the time format.

**VOLUME**

Returns the current volume setting. The volume is returned as a string in the format *left:right* where *left* and *right* are percentages of the maximum achievable effect for the left and right channels respectively. Leading zeros are suppressed for the volume level in each channel.

**WAIT**

The command is executed synchronously. The function waits until the requested action is complete before returning to the application. The WAIT keyword must be specified in order to receive return string information.

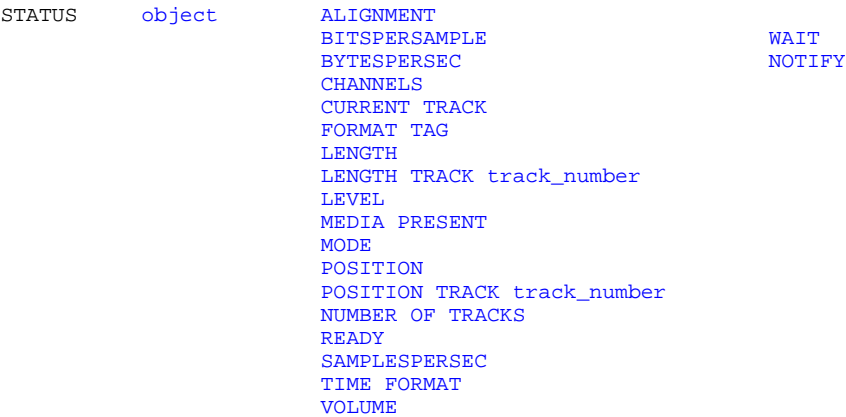
**NOTIFY**

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action

is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## STATUS (Waveaudio Command) - Syntax Diagram



### Examples

---

## STATUS (Waveaudio Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

---

## UNDO

---

## UNDO (Waveaudio Command) - Example

undo waveaudio wait

---

## UNDO (Waveaudio Command) - Purpose

The UNDO command undoes the last change to a file. The position of the media after the undo is 0.

---

## UNDO (Waveaudio Command) Keyword - object

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
  - Device name
  - Filename
  - Alias
- 

## UNDO (Waveaudio Command) Keyword - WAIT

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

---

## UNDO (Waveaudio Command) Keyword - NOTIFY

### NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

---

## UNDO (Waveaudio Command) - Keywords

### object

Object associated with this media control interface command. The object can be one of the following:

- Device type
- Device name
- Filename
- Alias

### WAIT

The command is executed synchronously. The function waits until the requested action is complete before returning to the application.

NOTIFY

The command is executed asynchronously, allowing control to be returned immediately to the application. When the requested action is complete, an [MM\\_MCINOTIFY](#) message is sent to the application window procedure.

UNDO (Waveaudio Command) - Syntax Diagram

UNDO      [object](#)

[WAIT](#)  
[NOTIFY](#)



UNDO (Waveaudio Command) - Topics

- Select an item:
- [Purpose](#)
  - [Syntax Diagram](#)
  - [Keywords](#)
  - [Example](#)
  - [Glossary](#)

Memory Playlist Commands

A memory playlist is a data structure in your application. It contains an array of simple, machine-like instructions, or commands, each of which has a fixed format consisting of a 32-bit operation code and three 32-bit operands.

Using playlist instructions, you can play audio objects in succession from one or more memory buffers. Instructions include branching to and returning from subroutines within the playlist. In addition, the playlist can be modified dynamically by the application while it is being played.

The MCI\_OPEN\_PLAYLIST flag is specified for the [MCI\\_OPEN](#) command message to indicate that the *pszElementName* field in the [MCI\\_OPEN\\_PARMS](#) data structure is a pointer to a memory playlist. The following table lists and describes the playlist instructions.

Command	Description
BRANCH_OPERATION	Transfers control to another instruction in the playlist.
CALL_OPERATION	Transfers control to the instruction specified in Operand 2, saving the number of the instruction following the CALL_OPERATION for use on a RETURN_OPERATION.
CUEPOINT_OPERATION	Causes a cue-point data record to be entered into the data stream.
DATA_OPERATION	Specifies a data buffer to be played



	from or recorded into.
EXIT_OPERATION	Indicates the end of the playlist.
LOOP_OPERATION	Controls iteration in a playlist.
MESSAGE_OPERATION	Returns a message to the application during playlist processing. MESSAGE_OPERATION statements can be used by the application to trace specific points during the execution of the playlist processor.
NOP_OPERATION	Used as a placeholder.
RETURN_OPERATION	Transfers control to the playlist instruction following the most recently executed CALL_OPERATION.
SEMPHORE_OPERATION	Causes the playlist processor to post an event semaphore. The playlist processor will call DosWaitEventSem.
SEMPHORE_WAIT_OPERATION	Causes the playlist processor to wait on a semaphore. The playlist processor will call DosWaitEventSem.

## Playlist Instructions

The commands and their descriptions (including operand information) follow:

### BRANCH\_OPERATION

Transfers control to another instruction in the playlist.

Operand 1	Ignored.
Operand 2	The absolute instruction number in the playlist to which control is being transferred. Because the playlist is defined as an array of structures (instruction, operation, and operand values) its first instruction is referenced as array element, index 0. Therefore, the first instruction in the list is 0, the second instruction is 1, and so on.
Operand 3	Ignored.

Branching out of a subroutine is not prohibited; however, it is not recommended because an unused return address is left on the stack maintained by the playlist processor.

An application can enable or disable a BRANCH\_OPERATION by exchanging it with a NOP\_OPERATION. Operands for a NOP\_OPERATION are ignored.

### CALL\_OPERATION

Transfers control to the absolute instruction number specified in Operand 2, saving the number of the instruction following the CALL for use on a RETURN instruction.

CALL instructions may be nested up to 20 levels.

Operand 1	Ignored.
Operand 2	Absolute instruction number in the playlist to which control is being transferred.
Operand 3	Ignored.

### CUEPOINT\_OPERATION

Causes a cue-point data record to be entered into the data stream. Note that the cue point is relative to the DATA\_OPERATION that follows it.

Operand 1	User-defined parameter to be returned as the low word of MsgParam1 in the <a href="#">MM_MCICUEPOINT</a> message.
Operand 2	Offset in MMTIME units for the actual time the CUEPOINT message should be generated.
Operand 3	Ignored.

The [MM\\_MCICUEPOINT](#) message is returned to the application as soon as possible after the cue-point data record is encountered in the data stream. The message is sent to the window handle specified when the device was originally opened.

**Note:** The CUEPOINT instruction is ignored when used in a recording operation.

#### DATA\_OPERATION

Specifies a data buffer to be played from or recorded into.

Operand 1	long pointer to a buffer in the application.
Operand 2	Length of the buffer pointed to by Operand 1.
Operand 3	Current position in the buffer. This operand is updated by the system during a recording or playback operation. For a playback operation, it is the number of bytes that have been sent to the output device handler. For a recording operation, it is the number of bytes that have been placed into a user buffer.
	The current position in the buffer is particularly important after a recording operation, because this field contains the number of bytes of recorded data. The remaining bytes in the buffer are not valid.

The buffer indicated by the DATA instruction must only contain the raw data bytes from the device and cannot include any header information. For example, a buffer for a sequencer device can contain only MIDI multibyte messages, as defined by the International MIDI Association. Therefore, the precise meaning or format of the data is dependent on the current settings of the media device. For example, a wave audio data element is assumed to have the format PCM or ADPCM, number of bits per sample, and so on, that is indicated by the settings of the audio device.

The address range of a DATA statement cannot overlap the address range of any another DATA statement. However, the same DATA statement can be repeated.

#### EXIT\_OPERATION

Indicates the end of the playlist.

Operand 1	Ignored.
Operand 2	Ignored.
Operand 3	Ignored.

#### LOOP\_OPERATION

Controls iteration in a playlist. It is the responsibility of the application to initialize the current iteration. The current iteration is reset to zero following loop termination.

Operand 1	Number of times the loop is to be executed.
Operand 2	Target instruction to branch to, when the loop condition fails.
Operand 3	Current iteration.

The last instruction in a loop is a branch back to the LOOP\_OPERATION. The operation of the LOOP\_OPERATION instruction is as follows:

1. If Operand 3 is less than Operand 1, control is transferred to the playlist instruction following the LOOP instruction, and the iteration count in Operand 3 is incremented.
2. Otherwise, the iteration count is reset to zero and control is passed to the instruction specified in Operand 2.

Typically, the application sets the iteration count to zero when the playlist is passed to the device, but this is not required. The loop instruction merely compares the loop count with the iteration count. If the iteration count is set to a value other than zero when the playlist is passed in, it is as if the loop has been executed that number of times. Also, if a playback operation is stopped, and then the same playlist is loaded again, the loop iteration count is not initialized by the playlist processor.

It is the application's responsibility to see that iteration count values are what is required when switching from play to record, record to play, and when changing settings for the data (for example, *bitspersample*, *samplespersec*, and so on) with the set command. These commands cause the playlist stream to be destroyed and re-created, and the playlist to be reassociated as a new playlist with the playlist processor.

#### MESSAGE\_OPERATION

Returns a message to the application during playlist processing.

Operand 1	Ignored.
Operand 2	ULONG that is returned to the application in the <a href="#">MM_MCIPLAYLISTMESSAGE</a> message MsgParam2.

Operand 3 Ignored.

Each time the playlist processor encounters a MESSAGE instruction, [MM\\_MCIPLAYLISTMESSAGE](#) is returned to the application. MESSAGE instructions can be used by the application to trace specific points during the execution of the playlist processor.

**Note:** This function is not intended to be used for timing of data production or consumption identified by previously interpreted instructions.

#### **NOP\_OPERATION**

Used as a placeholder.

Operand 1	Ignored.
Operand 2	Ignored.
Operand 3	Ignored.

#### **RETURN\_OPERATION**

Transfers control to the playlist instruction following the most recently executed CALL instruction.

Operand 1	Ignored.
Operand 2	Ignored.
Operand 3	Ignored.

#### **SEMPST\_OPERATION**

Causes the playlist processor to post an event semaphore. The playlist processor will call DosWaitEventSem.

Operand 1	Contains the semaphore to post.
Operand 2	Ignored.
Operand 3	Ignored.

#### **SEMWAIT\_OPERATION**

Causes the playlist processor to wait on a semaphore. The playlist processor will call DosWaitEventSem.

Operand 1	Contains the semaphore to perform the wait on.
Operand 2	Amount of time the semaphore should wait.
Operand 3	Boolean value indicating whether or not the semaphore should be cleared before waiting.

-----

## Graphic Button Control

This section describes graphic button styles and control messages. Graphic buttons are different from other types of push buttons in that they have graphic, two-state, and animation capabilities. Using graphic buttons, an application programmer can do the following:

- Display both text and bit maps on the same button.
- Animate a series of bit maps.
- Give the user a two-state button.
- Change bit maps.

Graphic buttons permit a more intuitive interaction between a user and an application than standard push buttons do. For example, graphic buttons can mimic and enhance the look and feel of the physical controls found on stereo equipment. They can emulate the two-state behavior of the Pause button found on cassette decks; scan play buttons on CD players and VCRs; and change or animate bit maps when control buttons are pressed.

The graphic button class is registered by calling [WinRegisterGraphicButton](#).

-----

## WinRegisterGraphicButton

-----

## WinRegisterGraphicButton - Syntax

This function registers the graphic button window class.

```
#define INCL_SW
#include <os2.h>

BOOL rc;

rc = WinRegisterGraphicButton();
```

---

## WinRegisterGraphicButton Return Value - rc

rc (BOOL) - returns

The following are return codes indicating success or failure:

TRUE	Success.
FALSE	Failure or not recognized.

---

## WinRegisterGraphicButton - Parameters

rc (BOOL) - returns

The following are return codes indicating success or failure:

TRUE	Success.
FALSE	Failure or not recognized.

---

## WinRegisterGraphicButton - Example Code

The following code illustrates registering a graphic button window class.

```
#define INCL_SW
#include <os2me.h>

HWND hwndFrame;

WinRegisterGraphicButton();          /* Register a graphic button control */

                                /* Create a Secondary Window */
hwndFrame = WinLoadSecondaryWindow(HWND_DESKTOP,
                                HWND_DESKTOP,
                                MyDlgProc,
                                NULL,
                                ID_DIALOG,
                                NULL);
```

# WinRegisterGraphicButton - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Example Code](#)  
[Glossary](#)

---

## Control Data

The [GBTNCDATA](#) data structure contains the information used as the graphic button control data.

---

## Styles

The base graphic button provides the ability to display bitmaps and text; it defaults to the Up state. Additional styles can be used to increase its function.

Graphic button controls have the following window styles:

GBS\_TWOSTATE

A two-state graphic button has two states-Up and Down. If the button is in the Up state, the button is drawn with the Z-order above the owner. If the button is in the Down state, the button is drawn with the Z-order below the owner. Typically, the Up state expresses to the user that the button's action is selectable; while the Down state expresses that the action is being processed currently. Related messages are [GBM\\_SETSTATE](#), [GBM\\_QUERYSTATE](#), [GBM\\_SETBITMAPINDEX](#), and [GBM\\_QUERYBITMAPINDEX](#).

GBS\_AUTOTWOSTATE

An automatic two-state graphic button automatically toggles its state from Up to Down and from Down to Up whenever the user clicks on it. No message from the owner is required for the button to toggle its state. Related messages are [GBM\\_SETSTATE](#), [GBM\\_QUERYSTATE](#), [GBM\\_SETBITMAPINDEX](#), and [GBM\\_QUERYBITMAPINDEX](#).

If both GBS\_TWOSTATE and GBS\_AUTOTWOSTATE are set, GBS\_AUTOTWOSTATE takes precedence.

GBS\_ANIMATION

An animation graphic button can animate a series of bitmaps. The bitmaps are displayed in ascending sequential order. The series is treated as a circular list: when the last bitmap of the series is reached, the next bitmap displayed is the first in the series. Related messages are [GBM\\_ANIMATE](#), [GBM\\_QUERYANIMATIONACTIVE](#), [GBM\\_SETANIMATIONRATE](#), [GBM\\_QUERYANIMATIONRATE](#), [GBM\\_SETBITMAPINDEX](#), and [GBM\\_QUERYBITMAPINDEX](#).

GBS\_AUTOANIMATION

An automatic animation graphic button automatically toggles the animation from On to Off and from Off to On whenever the user clicks on it. Related messages are [GBM\\_ANIMATE](#), [GBM\\_QUERYANIMATIONACTIVE](#), [GBM\\_SETANIMATIONRATE](#), [GBM\\_QUERYANIMATIONRATE](#), [GBM\\_SETBITMAPINDEX](#), and [GBM\\_QUERYBITMAPINDEX](#).

If both GBS\_ANIMATION and GBS\_AUTOANIMATION are set, GBS\_AUTOANIMATION takes precedence.

GBS\_HILITEBITMAP

This style permits display of a different bitmap when the graphic button is in the highlighted paint state. The highlighted paint state occurs

	when the user presses the pointer button while over the graphic button or holds the spacebar down when the graphic button has the focus. This bitmap is set by the <a href="#">GBM_SETBITMAPINDEX</a> message with GB_HILITE as the specified index to change. Related messages are <a href="#">GBM_SETBITMAPINDEX</a> and <a href="#">GBM_QUERYBITMAPINDEX</a> .
GBS_DISABLEBITMAP	This style permits display of a different bitmap when the graphic button is in the disabled paint state. This bitmap is set by the <a href="#">GBM_SETBITMAPINDEX</a> message with GB_DISABLE as the specified index to change. Related messages are <a href="#">GBM_SETBITMAPINDEX</a> and <a href="#">GBM_QUERYBITMAPINDEX</a> .
GBS_3D_TEXTRECESSED	The text on the graphic button is displayed three-dimensionally. The Z-order of the text is below the face of the graphic button. The text is drawn with its main body black, and the bottom and right edges white.
GBS_3D_TEXTRAISED	The text on the graphic button is displayed three-dimensionally. The Z-order of the text is above the face of the graphic button. The text is drawn with its main body white, and the bottom and right edges black.
	If both GBS_3D_TEXTRECESSED and GBS_3D_TEXTRAISED are set, GBS_3D_TEXTRAISED takes precedence.

## Control Messages

Graphic buttons use the following messages:

Message	Description
<a href="#">GBM_ANIMATE</a>	Sets the animation of an animate-able graphic button On or Off.
<a href="#">GBM_QUERYANIMATIONACTIVE</a>	Returns the animation state of the graphic button.
<a href="#">GBM_QUERYANIMATIONRATE</a>	Returns the animation rate of a graphic button in milliseconds.
<a href="#">GBM_QUERYBITMAPINDEX</a>	Returns the bitmap index of the queried button parameter.
<a href="#">GBM_QUERYSTATE</a>	Returns the current state of the graphic button.
<a href="#">GBM_QUERYTEXTPOSITION</a>	Returns the text positioning relative to the bitmap.
<a href="#">GBM_SETANIMATIONRATE</a>	Sets the animation rate of a animate-able graphic button.
<a href="#">GBM_SETBITMAPINDEX</a>	Sets the bitmap index to use during various states of the graphic button.
<a href="#">GBM_SETGRAPHICDATA</a>	An application sends this message to change the graphical data of the graphic button.
<a href="#">GBM_SETSTATE</a>	Sets the state of a two-state graphic button.
<a href="#">GBM_SETTEXTPOSITION</a>	Sets the text positioning relative to the bitmap.

---

## WM\_COMMAND

---

### WM\_COMMAND Field - usCommand

**usCommand** ([USHORT](#))

The command value. The application must relate the command to an application function.

---

### WM\_COMMAND Field - usSource

**usSource** ([USHORT](#))

Identifies the type of control that generated this command as CMDSRC\_PUSHBUTTON.

---

### WM\_COMMAND Field - usPointer

**usPointer** ([USHORT](#))

The pointer-device indicator.

---

### WM\_COMMAND Return Value - rc

**rc** ([ULONG](#))

Reserved.

---

## WM\_COMMAND - Parameters

**usCommand** ([USHORT](#))

The command value. The application must relate the command to an application function.

**usSource** ([USHORT](#))  
Identifies the type of control that generated this command as CMDSRC\_PUSHBUTTON.

**usPointer** ([USHORT](#))  
The pointer-device indicator.

**rc** ([ULONG](#))  
Reserved.

---

## WM\_COMMAND - Description

This message occurs when the graphic button has a significant event to notify its owner.

```
param1
    USHORT  usCommand  /* Command value. */

param2
    USHORT  usSource    /* Source of command. */
    USHORT  usPointer   /* Pointer-device indicator. */
```

---

## WM\_COMMAND - Topics

- Select an item:
- [Description](#)
  - [Parameters](#)
  - [Returns](#)
  - [Glossary](#)
- 

## WM\_CONTROL

---

### WM\_CONTROL Field - usID

**usID** ([USHORT](#))  
The identity of the graphic button that generated the notification.

---

### WM\_CONTROL Field - usnotifycode



**usnotifycode** (USHORT)

The notification codes that indicate what action has occurred.

**GBN\_BUTTONUP**

Returned when the button is in the Up position. *param2* is reserved.

**GBN\_BUTTONDOWN**

Returned when the button is in the Down position. *param2* is reserved.

**GBN\_BUTTONHILITE**

Returned when the button is selected to the On position. *param2* is reserved.

**GBN\_SETFOCUS**

Returned when the button is gaining or losing focus. *param2* is reserved.

-----

## WM\_CONTROL Field - ulnotifyspec

**ulnotifyspec** (ULONG)

Notify command-specific information.

-----

## WM\_CONTROL Return Value - ulReserved

**ulReserved** (ULONG)

Reserved.

-----

## WM\_CONTROL - Parameters

**usID** (USHORT)

The identity of the graphic button that generated the notification.

**usnotifycode** (USHORT)

The notification codes that indicate what action has occurred.

**GBN\_BUTTONUP**

Returned when the button is in the Up position. *param2* is reserved.

**GBN\_BUTTONDOWN**

Returned when the button is in the Down position. *param2* is reserved.

**GBN\_BUTTONHILITE**

Returned when the button is selected to the On position. *param2* is reserved.

**GBN\_SETFOCUS**

Returned when the button is gaining or losing focus. *param2* is reserved.

**ulnotifyspec** (ULONG)

Notify command-specific information.

**ulReserved** (ULONG)

Reserved.

---

## WM\_CONTROL - Description

This message provides notification codes initiated by the graphic button control window to notify its owner of significant events.

```
param1
    USHORT  usID          /* Graphic button ID. */
    USHORT  usnotifycode /* Notification code. */

param2
    ULONG   ulnotifyspec /* Command-specific info. */
```

---

## WM\_CONTROL - Remarks

These notifications are generated by all graphic buttons. The *param2* field is reserved.

---

## WM\_CONTROL - Default Processing

None.

---

## WM\_CONTROL - Topics

Select an item:

- [Description](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Default Processing](#)
- [Glossary](#)

---

## GBM\_ANIMATE

---

## GBM\_ANIMATE Field - usStart

**usStart** ([USHORT](#))  
Start/stop animation.

- TRUE                      Start animating the graphic button.
- FALSE                     Stop animating the graphic button.

-----

## GBM\_ANIMATE Field - usContinue

**usContinue** ([USHORT](#))  
Starting point of animation.

- TRUE                      Animation starts with the currently shown bitmap. For example, if the graphic button is in the Down state, the animation begins with the Down bitmap and continues with subsequent bitmaps in the array until the end of the animation series. Then, the animation restarts at the beginning of the animation series (but not necessarily with the Down bitmap).
- FALSE                     Animation starts at the beginning of the animation bitmap series. The start of this series can be specified by a [GBM\\_SETBITMAPINDEX](#) message.

-----

## GBM\_ANIMATE Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure.

- TRUE                      Success.
- FALSE                     Failure or not recognized.

-----

## GBM\_ANIMATE - Parameters

**usStart** ([USHORT](#))  
Start/stop animation.

- TRUE                      Start animating the graphic button.
- FALSE                     Stop animating the graphic button.

**usContinue** ([USHORT](#))  
Starting point of animation.

TRUE	Animation starts with the currently shown bitmap. For example, if the graphic button is in the Down state, the animation begins with the Down bitmap and continues with subsequent bitmaps in the array until the end of the animation series. Then, the animation restarts at the beginning of the animation series (but not necessarily with the Down bitmap).
FALSE	Animation starts at the beginning of the animation bitmap series. The start of this series can be specified by a <a href="#">GBM_SETBITMAPINDEX</a> message.
<b>rc (ULONG)</b>	Return codes indicating success or failure.
TRUE	Success.
FALSE	Failure or not recognized.

-----

## GBM\_ANIMATE - Description

This message sets the animation of an animate-able graphic button to On or Off.

```
param1
    USHORT  usStart      /* Start/stop animation. */

param2
    USHORT  usContinue   /* Starting point of animation. */
```

-----

## GBM\_ANIMATE - Remarks

This message does not have any effect if the graphic button is not of style GBS\_ANIMATE or GBS\_AUTOANIMATE.

-----

## GBM\_ANIMATE - Topics

Select an item:

[Description](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Glossary](#)

-----

## GBM\_QUERYANIMATIONACTIVE

---

# GBM\_QUERYANIMATIONACTIVE Field - ulReserved

ulReserved (ULONG)  
Reserved.

---

# GBM\_QUERYANIMATIONACTIVE Field - ulReserved

ulReserved (ULONG)  
Reserved.

---

# GBM\_QUERYANIMATIONACTIVE Return Value - rc

rc (ULONG)  
Return codes indicating success or failure.

TRUE	Animation is active. The graphic button is currently animating.
FALSE	Animation is not active. The graphic button is not currently animating.

---

# GBM\_QUERYANIMATIONACTIVE - Parameters

ulReserved (ULONG)  
Reserved.

ulReserved (ULONG)  
Reserved.

rc (ULONG)  
Return codes indicating success or failure.

TRUE	Animation is active. The graphic button is currently animating.
FALSE	Animation is not active. The graphic button is not currently animating.

---

# GBM\_QUERYANIMATIONACTIVE - Description

This message returns the animation state of the graphic button.

```
param1
    ULONG ulReserved /* Reserved. */

param2
    ULONG ulReserved /* Reserved. */
```

---

## GBM\_QUERYANIMATIONACTIVE - Topics

- Select an item:
- [Description](#)
  - [Parameters](#)
  - [Returns](#)
  - [Glossary](#)

---

## GBM\_QUERYANIMATIONRATE

---

### GBM\_QUERYANIMATIONRATE Field - ulReserved

**ulReserved (ULONG)**  
Reserved.

---

### GBM\_QUERYANIMATIONRATE Field - ulReserved

**ulReserved (ULONG)**  
Reserved.

---

### GBM\_QUERYANIMATIONRATE Return Value - rc

**rc (ULONG)**  
The animation rate, in milliseconds, used for this graphic button.

---

# GBM\_QUERYANIMATIONRATE - Parameters

- ulReserved (ULONG)**  
Reserved.
- ulReserved (ULONG)**  
Reserved.
- rc (ULONG)**  
The animation rate, in milliseconds, used for this graphic button.

---

# GBM\_QUERYANIMATIONRATE - Description

This message returns the animation rate of the graphic button in milliseconds.

```
param1
    ULONG  ulReserved  /*  Reserved.  */

param2
    ULONG  ulReserved  /*  Reserved.  */
```

---

# GBM\_QUERYANIMATIONRATE - Topics

- Select an item:
- [Description](#)
  - [Parameters](#)
  - [Returns](#)
  - [Glossary](#)

---

# GBM\_QUERYBITMAPINDEX

---

# GBM\_QUERYBITMAPINDEX Field - usGBState

- usGBState (USHORT)**  
Bitmap state.
- GB\_UP

	Query bitmap used when in the Up state.
GB_DOWN	Query bitmap used when in the Down state.
GB_HILITE	Query bitmap used when in the highlighted state.
GB_DISABLE	Query bitmap used when in the disabled state.
GB_ANIMATIONBEGIN	Query bitmap at the beginning of an animation series.
GB_ANIMATIONEND	Query bitmap at the end of an animation series.
GB_CURRENTSTATE	Query the index of the currently shown bitmap.
GB_ERROR	Invalid parameter.

-----

## GBM\_QUERYBITMAPINDEX Field - ulReserved

**ulReserved** (ULONG)  
Reserved.

-----

## GBM\_QUERYBITMAPINDEX Return Value - rc

**rc** (USHORT)  
Bitmap index of the queried button parameter.

-----

## GBM\_QUERYBITMAPINDEX - Parameters

<b>usGBState</b> (USHORT)	Bitmap state.
GB_UP	Query bitmap used when in the Up state.
GB_DOWN	Query bitmap used when in the Down state.
GB_HILITE	Query bitmap used when in the highlighted state.
GB_DISABLE	Query bitmap used when in the disabled state.
GB_ANIMATIONBEGIN	



Query bitmap at the beginning of an animation series.

GB\_ANIMATIONEND  
Query bitmap at the end of an animation series.

GB\_CURRENTSTATE  
Query the index of the currently shown bitmap.

GB\_ERROR  
Invalid parameter.

**ulReserved** ([ULONG](#))  
Reserved.

**rc** ([USHORT](#))  
Bitmap index of the queried button parameter.

---

## GBM\_QUERYBITMAPINDEX - Description

This message returns the bitmap index of the queried button parameter.

```
param1
    USHORT  usGBState /* Bitmap state. */

param2
    ULONG   ulReserved /* Reserved. */
```

---

## GBM\_QUERYBITMAPINDEX - Remarks

If *usGBState* is not valid, then the index of the currently shown bitmap is returned.

---

## GBM\_QUERYBITMAPINDEX - Topics

Select an item:

- [Description](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

---

## GBM\_QUERYSTATE

---

# GBM\_QUERYSTATE Field - ulReserved

**ulReserved (ULONG)**  
Reserved.

-----

# GBM\_QUERYSTATE Field - ulReserved

**ulReserved (ULONG)**  
Reserved.

-----

# GBM\_QUERYSTATE Return Value - rc

**rc (ULONG)**  
Return codes indicating the state of the graphic button.

- GB\_UP                   The graphic button is in the Up state.
- GB\_DOWN               The graphic button is in the Down state.
- GB\_HILITE             The graphic button is in the highlighted state.
- GB\_ERROR              An error has occurred.

-----

# GBM\_QUERYSTATE - Parameters

**ulReserved (ULONG)**  
Reserved.

**ulReserved (ULONG)**  
Reserved.

**rc (ULONG)**  
Return codes indicating the state of the graphic button.

- GB\_UP                   The graphic button is in the Up state.
- GB\_DOWN               The graphic button is in the Down state.
- GB\_HILITE             The graphic button is in the highlighted state.

GB\_ERROR

An error has occurred.

-----

## GBM\_QUERYSTATE - Description

This message returns the current state of the graphic button.

```
param1
    ULONG   ulReserved /* Reserved. */

param2
    ULONG   ulReserved /* Reserved. */
```

-----

## GBM\_QUERYSTATE - Remarks

Graphic buttons that are not of either GBS\_TWOSTATE or GBS\_AUTOTWOSTATE styles always are considered in the Up state.

Animation can occur independently from a change in state of the button. The drawing of animation bitmaps takes precedence over the current state bitmap.

A multi-state button or a button that draws bitmaps in its highlighted or disabled states is not intended to be animated, and animated buttons are not intended to have visual states. The combination of these two styles might yield undefined results.

-----

## GBM\_QUERYSTATE - Topics

Select an item:

[Description](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Glossary](#)

-----

## GBM\_QUERYTEXTPOSITION

-----

## GBM\_QUERYTEXTPOSITION Field - ulReserved

**ulReserved** (ULONG)  
Reserved.

---

## GBM\_QUERYTEXTPOSITION Field - ulReserved

**ulReserved** (ULONG)  
Reserved.

---

## GBM\_QUERYTEXTPOSITION Return Value - rc

**rc** (ULONG)  
Return codes indicating the position of the graphic button.

GB\_TEXTBELOW  
The graphic button text is located below the bitmap.

GB\_TEXTABOVE  
The graphic button text is located above the bitmap.

---

## GBM\_QUERYTEXTPOSITION - Parameters

**ulReserved** (ULONG)  
Reserved.

**ulReserved** (ULONG)  
Reserved.

**rc** (ULONG)  
Return codes indicating the position of the graphic button.

GB\_TEXTBELOW  
The graphic button text is located below the bitmap.

GB\_TEXTABOVE  
The graphic button text is located above the bitmap.

---

## GBM\_QUERYTEXTPOSITION - Description

This message returns text positioning relative to the bitmap.

```
param1
    ULONG ulReserved /* Reserved. */
```

```
param2
    ULONG ulReserved /* Reserved. */
```

## GBM\_QUERYTEXTPOSITION - Topics

- Select an item:
- Description
  - Parameters
  - Returns
  - Glossary

## GBM\_SETANIMATIONRATE

### GBM\_SETANIMATIONRATE Field - ulMil

**ulMil** (**ULONG**)  
The rate at which bit maps are displayed from the animation series. The rate is specified as the period between bit-map updates in milliseconds.

### GBM\_SETANIMATIONRATE Field - ulReserved

**ulReserved** (**ULONG**)  
Reserved.

### GBM\_SETANIMATIONRATE Return Value - rc

- rc** (**ULONG**)  
Return code indicating success or failure.
- |       |                            |
|-------|----------------------------|
| TRUE  | Success.                   |
| FALSE | Failure or not recognized. |

---

## GBM\_SETANIMATIONRATE - Parameters

### **ulMil** ([ULONG](#))

The rate at which bit maps are displayed from the animation series. The rate is specified as the period between bit-map updates in milliseconds.

### **ulReserved** ([ULONG](#))

Reserved.

### **rc** ([ULONG](#))

Return code indicating success or failure.

TRUE

Success.

FALSE

Failure or not recognized.

---

## GBM\_SETANIMATIONRATE - Description

This message sets the animation rate of an animate-able graphic button.

```
param1
    ULONG  ulMil      /* Rate at which bit maps are displayed. */
param2
    ULONG  ulReserved /* Reserved. */
```

---

## GBM\_SETANIMATIONRATE - Remarks

This message does not have any effect if the graphic button is not of GBS\_ANIMATE or GBS\_AUTOANIMATE styles.

---

## GBM\_SETANIMATIONRATE - Topics

Select an item:

[Description](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Glossary](#)

---

## GBM\_SETBITMAPINDEX

---

## GBM\_SETBITMAPINDEX Field - usGBState

### usGBState (USHORT)

State of graphic button whose index is to be changed.

GB_UP	
GB_DOWN	Set the bitmap used when in the Up state.
GB_HILITE	Set the bitmap used when in the Down state.
GB_DISABLE	Set the bitmap used when in the highlighted state.
GB_ANIMATIONBEGIN	Set the bitmap used when in the disabled state.
GB_ANIMATIONEND	Where animation must begin.
GB_CURRENTSTATE	Where animation must end.
	Refers to either up or down bitmaps; depends on whether the button is in the Up or Down state.

---

## GBM\_SETBITMAPINDEX Field - usFrameCode

### usFrameCode (USHORT)

Frame code.

GB_INDEX_FORWARD	
GB_INDEX_BACKWARD	Advance to next bitmap index in circular array.
GB_INDEX_FIRST	Set to previous bitmap index in circular array.
GB_INDEX_LAST	Set to first bitmap index.
desired_bit_map	Set to last bitmap index.
	Otherwise, use <i>usFrameCode</i> number as the bitmap index.

---

## GBM\_SETBITMAPINDEX Return Value - rc

### rc (ULONG)

The return code indicating success or failure.

TRUE	
FALSE	Success.
	Failure or not recognized.

---

# GBM\_SETBITMAPINDEX - Parameters

## usGBState (USHORT)

State of graphic button whose index is to be changed.

GB_UP	Set the bitmap used when in the Up state.
GB_DOWN	Set the bitmap used when in the Down state.
GB_HILITE	Set the bitmap used when in the highlighted state.
GB_DISABLE	Set the bitmap used when in the disabled state.
GB_ANIMATIONBEGIN	Where animation must begin.
GB_ANIMATIONEND	Where animation must end.
GB_CURRENTSTATE	Refers to either up or down bitmaps; depends on whether the button is in the Up or Down state.

## usFrameCode (USHORT)

Frame code.

GB_INDEX_FORWARD	Advance to next bitmap index in circular array.
GB_INDEX_BACKWARD	Set to previous bitmap index in circular array.
GB_INDEX_FIRST	Set to first bitmap index.
GB_INDEX_LAST	Set to last bitmap index.
desired_bit_map	Otherwise, use <i>usFrameCode</i> number as the bitmap index.

## rc (ULONG)

The return code indicating success or failure.

TRUE	Success.
FALSE	Failure or not recognized.

-----

# GBM\_SETBITMAPINDEX - Description

This message sets the bitmap index to use during various states of the graphic button.

```
param1
    USHORT  usGBState    /* State of graphic button whose index is to be changed. */

param2
    USHORT  usFrameCode  /* Frame code. */
```

-----

# GBM\_SETBITMAPINDEX - Remarks



If starting, the animation bit should be less than or equal to the ending animation bitmap index.  
Setting the bitmap index for unused states is undefined.

---

## GBM\_SETBITMAPINDEX - Topics

- Select an item:
- [Description](#)
  - [Parameters](#)
  - [Returns](#)
  - [Remarks](#)
  - [Glossary](#)

---

## GBM\_SETGRAPHICDATA

---

### GBM\_SETGRAPHICDATA Field - gbtnpdata

**gbtnpdata** ([PGBTNCDATA](#))  
New graphic button data to use.

---

### GBM\_SETGRAPHICDATA Field - ulReserved

**ulReserved** ([ULONG](#))  
Reserved.

---

### GBM\_SETGRAPHICDATA Return Value - rc

**rc** ([ULONG](#))  
Return code indicating success or failure.

TRUE	Success.
FALSE	Failure or not recognized.

---

## GBM\_SETGRAPHICDATA - Parameters

**gbtncdata** ([PGBTNCDATA](#))

New graphic button data to use.

**ulReserved** ([ULONG](#))

Reserved.

**rc** ([ULONG](#))

Return code indicating success or failure.

TRUE

Success.

FALSE

Failure or not recognized.

---

## GBM\_SETGRAPHICDATA - Description

An application sends this message to change the graphical data of the graphic button. This data includes the text and bit maps.

```
param1
    PGBTNCDATA  gbtncdata    /* New graphic button data to use. */

param2
    ULONG       ulReserved   /* Reserved. */
```

---

## GBM\_SETGRAPHICDATA - Remarks

Setting the graphic button data with this message erases all previous data related to the state of the graphic button. The state of the graphic button will be set to the default parameters.

WinSetWindowText can be used to change the text of the graphic button without altering the button's parameters or state.

---

## GBM\_SETGRAPHICDATA - Example Code

The following code illustrates how to change graphical data for a graphic button.

```
typedef struct {
    USHORT  usReserved;           /*Reserved field */
    PSZ     pszText;             /*Initial text of button */
    HMODULE  hmod;               /*Module handle for bit maps */
    USHORT  cBitmaps;            /*Number of bit maps */
    USHORT  aidBitmap[1];        /*Array of bit-map IDs */
}GBTNCDATA;
typedef GBTNCDATA *PGBTNCDATA;
```

```

GBTNCDATA  cdata;

cdata.usReserved = GB_STRUCTURE;
cdata.pszText    = "Play";
cdata.hmod       = hmodBitmaps;
cdata.cBitmaps   = 1;
cdata.aidBitmap[0] = ID_PLAY;

WinSendMsg ( hwndButton, GBM_SETGRAPHICDATA, (MPARAM)&cdata, 0L;

```

---

## GBM\_SETGRAPHICDATA - Topics

Select an item:

[Description](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Glossary](#)

---

## GBM\_SETSTATE

---

### GBM\_SETSTATE Field - usStateCode

**usStateCode** ([USHORT](#))

State code.

GB\_UP

Sets the graphic button to the Up state.

GB\_DOWN

Sets the graphic button to the Down state.

GB\_TOGGLE

Toggles the graphic buttons state from Up to Down or from Down to Up.

---

### GBM\_SETSTATE Field - usRePaint

**usRePaint** ([USHORT](#))

Indicates when change of state will occur.

TRUE

The change in state is reflected immediately onscreen.

FALSE

The change in state is not reflected immediately onscreen. The graphic button displays the change the next time it

receives a BN\_PAINT message.

---

## GBM\_SETSTATE Return Value - rc

**rc** (ULONG)

Return codes indicating success or failure.

TRUE

Success.

FALSE

Failure or not recognized.

---

## GBM\_SETSTATE - Parameters

**usStateCode** (USHORT)

State code.

GB\_UP

Sets the graphic button to the Up state.

GB\_DOWN

Sets the graphic button to the Down state.

GB\_TOGGLE

Toggles the graphic buttons state from Up to Down or from Down to Up.

**usRePaint** (USHORT)

Indicates when change of state will occur.

TRUE

The change in state is reflected immediately onscreen.

FALSE

The change in state is not reflected immediately onscreen. The graphic button displays the change the next time it receives a BN\_PAINT message.

**rc** (ULONG)

Return codes indicating success or failure.

TRUE

Success.

FALSE

Failure or not recognized.

---

## GBM\_SETSTATE - Description

This message sets the state of a two-state graphic button.

```
param1
    USHORT  usStateCode  /* State code. */
```

```
param2
```

```
USHORT  usRePaint    /* Indicates when change of state will occur. */
```

---

# GBM\_SETSTATE - Topics

- Select an item:
- [Description](#)
  - [Parameters](#)
  - [Returns](#)
  - [Glossary](#)

---

# GBM\_SETTEXTPOSITION

---

## GBM\_SETTEXTPOSITION Field - usTextPos

- usTextPos** ([USHORT](#))  
Text position.
- |              |                                 |
|--------------|---------------------------------|
| GB_TEXTBELOW | Position text below the bitmap. |
| GB_TEXTABOVE | Position text above the bitmap. |

---

## GBM\_SETTEXTPOSITION Field - ulReserved

- ulReserved** ([ULONG](#))  
Reserved.

---

## GBM\_SETTEXTPOSITION Return Value - rc

- rc** ([USHORT](#))  
Return codes indicating success or failure.
- |       |                            |
|-------|----------------------------|
| TRUE  | Success.                   |
| FALSE | Failure or not recognized. |

---

## GBM\_SETTEXTPOSITION - Parameters

**usTextPos** ([USHORT](#))

Text position.

GB\_TEXTBELOW

Position text below the bitmap.

GB\_TEXTABOVE

Position text above the bitmap.

**ulReserved** ([ULONG](#))

Reserved.

**rc** ([USHORT](#))

Return codes indicating success or failure.

TRUE

Success.

FALSE

Failure or not recognized.

---

## GBM\_SETTEXTPOSITION - Description

This message sets the text positioning relative to the bitmap.

```
param1
    USHORT  usTextPos    /* Text position. */

param2
    ULONG   ulReserved   /* Reserved. */
```

---

## GBM\_SETTEXTPOSITION - Remarks

The default text position is below the bitmap. If this graphic button has a null text string, then this message has no visual effect.

---

## GBM\_SETTEXTPOSITION - Topics

Select an item:

[Description](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

---

## Secondary Window Functions

The secondary window manager provides all the capability of a standard dialog window, but it permits the window to be sizable and automatically formats the window and manages the scroll bars.

The functions that manipulate and control the secondary window are designed to convert easily from the standard dialog manager.

This section describes the secondary window functions and data structures.

The following table lists the secondary window functions.

Function	Description
<a href="#">WinCreateSecondaryWindow</a>	Creates a secondary window from a dialog template in memory.
<a href="#">WinDefSecondaryWindowProc</a>	Provides the default behavior for a secondary window.
<a href="#">WinDefaultSize</a>	Sizes the window to its default size-the optimal size of the dialog window.
<a href="#">WinDestroySecondaryWindow</a>	Eliminates a secondary window.
<a href="#">WinDismissSecondaryWindow</a>	Causes modal WinProcessSecondaryWindow or WinSecondaryWindow calls to return.
<a href="#">WinInsertDefaultSize</a>	Adds the default size function to the system menu of a secondary window.
<a href="#">WinLoadSecondaryWindow</a>	Creates a secondary window from a dialog template in a resource.
<a href="#">WinProcessSecondaryWindow</a>	Processes a modal secondary window by dispatching messages while the modal window is displayed.
<a href="#">WinQuerySecondaryHWND</a>	Returns a window handle to various windows created as part of a secondary window.
<a href="#">WinSecondaryMessageBox</a>	Creates a modal window similar to WinMessageBox that can be used to display error messages and ask questions.
<a href="#">WinSecondaryWindow</a>	Loads and processes a modal secondary window and returns the result value established by WinDismissSecondaryWindow.

---

## WM\_INITSECONDARYWINDOW

---

## WM\_INITSECONDARYWINDOW Field - ulReserved

**ulReserved** ([ULONG](#))

Reserved value. Set to zero.

---

## WM\_INITSECONDARYWINDOW Field - pcreate

**pcreate** ([PVOID](#))

This points to the data area that is passed by the [WinLoadSecondaryWindow](#), [WinCreateSecondaryWindow](#), and [WinSecondaryWindow](#) functions in their *pCreateParams* parameter.

---

## WM\_INITSECONDARYWINDOW Return Value - ulReserved

**ulReserved** ([ULONG](#))

Reserved value. Set to zero.

---

## WM\_INITSECONDARYWINDOW - Parameters

**ulReserved** ([ULONG](#))

Reserved value. Set to zero.

**pcreate** ([PVOID](#))

This points to the data area that is passed by the [WinLoadSecondaryWindow](#), [WinCreateSecondaryWindow](#), and [WinSecondaryWindow](#) functions in their *pCreateParams* parameter.

**ulReserved** ([ULONG](#))

Reserved value. Set to zero.

---

## WM\_INITSECONDARYWINDOW - Description

This message occurs when a secondary window is being created.

```
param1
    ULONG ulReserved /* Reserved. */

param2
    PVOID pcreate /* Application-defined data area. */
```



---

## WM\_INITSECONDARYWINDOW - Remarks

This message occurs after the WM\_INITDLG message and after the secondary window is set to its initial size. Application sizing adjustments should be made with this message, rather than with WM\_INITDLG.

---

## WM\_INITSECONDARYWINDOW - Related Messages

- [WM\\_INITDLG](#)
- 

## WM\_INITSECONDARYWINDOW - Topics

Select an item:

[Description](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Related Messages](#)

[Glossary](#)

---

## WinCreateSecondaryWindow

---

## WinCreateSecondaryWindow - Syntax

This function creates a secondary window from a dialog template in memory.

```
#define INCL_SW
#include <os2.h>

HWND      hwndParent;      /* Parent window handle. */
HWND      hwndOwner;       /* Window-handle owner. */
PFNWP     pfnDlgProc;      /* Secondary window procedure. */
PDLGTEMPLATE pdlgt;        /* Dialog template. */
PVOID     pCreateParams;   /* Parameters passed to secondary window proc. */
HWND      hwnd;            /* Window handle. */

hwnd = WinCreateSecondaryWindow(hwndParent,
                                hwndOwner, pfnDlgProc, pdlgt, pCreateParams);
```

-----

## WinCreateSecondaryWindow Parameter - hwndParent

**hwndParent** ([HWND](#)) - input  
The parent window handle of the secondary window to be created.

-----

## WinCreateSecondaryWindow Parameter - hwndOwner

**hwndOwner** ([HWND](#)) - input  
The owner-window handle of the secondary window to be created.

-----

## WinCreateSecondaryWindow Parameter - pfnDlgProc

**pfnDlgProc** ([PFNWNDP](#)) - input  
The secondary window procedure.

-----

## WinCreateSecondaryWindow Parameter - pDlgT

**pDlgT** ([PDLGTEMPLATE](#)) - input  
The dialog template that defines the layout of the window.

-----

## WinCreateSecondaryWindow Parameter - pCreateParams

**pCreateParams** ([PVOID](#)) - input  
The parameters passed to the secondary window procedure on the WM\_INITDLG message.

-----

## WinCreateSecondaryWindow Return Value - hwnd

**hwnd** ([HWND](#)) - returns

Returns the frame window handle a secondary window.

---

## WinCreateSecondaryWindow - Parameters

**hwndParent** ([HWND](#)) - input

The parent window handle of the secondary window to be created.

**hwndOwner** ([HWND](#)) - input

The owner-window handle of the secondary window to be created.

**pfnDlgProc** ([PFNWP](#)) - input

The secondary window procedure.

**pdlgt** ([PDLGTEMPLATE](#)) - input

The dialog template that defines the layout of the window.

**pCreateParams** ([PVOID](#)) - input

The parameters passed to the secondary window procedure on the WM\_INITDLG message.

**hwnd** ([HWND](#)) - returns

Returns the frame window handle a secondary window.

---

## WinCreateSecondaryWindow - Remarks

Refer to WinCreateDlg in the *IBM OS/2 Presentation Manager Programming Reference*.

---

## WinCreateSecondaryWindow - Related Functions

- [WinDismissSecondaryWindow](#)
  - [WinSecondaryWindow](#)
  - [WinLoadSecondaryWindow](#)
  - [WinProcessSecondaryWindow](#)
- 

## WinCreateSecondaryWindow - Example Code

The following code illustrates how create a secondary window from a dialog template in memory.

```
#define INCL_SW
#include <os2me.h>

PDLGTEMPLATE pdlgt;

                hmod (modulehandle)
DosGetResource (NULL, RT_DIALOG, ID_DIALOG, (PVOID) pdlgt);

WinCreateSecondaryWindow ( HWND_DESKTOP, /* Parent window      */ */
```

```
HwndOwner          /* Owner window      */
MyDlgProc,         /* Dialog procedure */
pdlgt,             /* Dialog template  */
NULL);            /* Create parameters */
```

---

## WinCreateSecondaryWindow - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

## WinDefSecondaryWindowProc

---

## WinDefSecondaryWindowProc - Syntax

This function provides the default behavior for a secondary window.

```
#define INCL_SW
#include <os2.h>

HWND      hwnd; /* Dialog window handle. */
ULONG     msg;  /* Message identity. */
MPARAM    mp1;  /* Parameter 1. */
MPARAM    mp2;  /* Parameter 2. */
MRESULT   rc;   /* Returns MRESULT. */

rc = WinDefSecondaryWindowProc(hwnd, msg,
                               mp1, mp2);
```

---

## WinDefSecondaryWindowProc Parameter - hwnd

**hwnd** (**HWND**) - input  
Dialog window handle.

---

## WinDefSecondaryWindowProc Parameter - msg

**msg** (**ULONG**) - input  
Message identity.

---

## WinDefSecondaryWindowProc Parameter - mp1

**mp1** (**MPARAM**) - input  
Parameter 1.

---

## WinDefSecondaryWindowProc Parameter - mp2

**mp2** (**MPARAM**) - input  
Parameter 2.

---

## WinDefSecondaryWindowProc Return Value - rc

**rc** (**MRESULT**) - returns  
Returns the HRESULT defined by the message passed to this function.

---

## WinDefSecondaryWindowProc - Parameters

**hwnd** (**HWND**) - input  
Dialog window handle.

**msg** (**ULONG**) - input  
Message identity.

**mp1** (**MPARAM**) - input  
Parameter 1.

**mp2** (**MPARAM**) - input  
Parameter 2.

**rc** (**MRESULT**) - returns  
Returns the HRESULT defined by the message passed to this function.

---

## WinDefSecondaryWindowProc - Remarks

A secondary window procedure must reference this function for messages that are not handled explicitly. Refer to WinDefDlgProc in the *IBM OS/2 Presentation Manager Programming Reference*.

---

## WinDefSecondaryWindowProc - Related Functions

- [WinCreateSecondaryWindow](#)
- [WinDismissSecondaryWindow](#)
- [WinSecondaryWindow](#)
- [WinLoadSecondaryWindow](#)
- [WinProcessSecondaryWindow](#)

---

## WinDefSecondaryWindowProc - Example Code

The following code illustrates how to provide default behavior for a secondary window.

```
#define INCL_SW
#include <os2me.h>

MRESULT MyDlgProc (HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
    switch (msg) {

    }

    /* Call WinDefSecondaryWindowProc for default processing */
    return (WinDefSecondaryWindowProc (hwnd, msg, mp1, mp2));
}
```

---

## WinDefSecondaryWindowProc - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## WinDefaultSize

---

# WinDefaultSize - Syntax

This function sizes the window to its default size-the optimal size of the dialog window.

```
#define INCL_SW
#include <os2.h>

HWND    hwnd; /* Secondary window handle. */
BOOL    rc;    /* Return codes. */

rc = WinDefaultSize(hwnd);
```

---

## WinDefaultSize Parameter - hwnd

hwnd (HWND) - input  
Secondary window handle.

---

## WinDefaultSize Return Value - rc

rc (BOOL) - returns  
Return codes indicating success or failure:

TRUE	Success.
FALSE	Failure or not recognized.

---

## WinDefaultSize - Parameters

hwnd (HWND) - input  
Secondary window handle.

rc (BOOL) - returns  
Return codes indicating success or failure:

TRUE	Success.
FALSE	Failure or not recognized.

---

## WinDefaultSize - Remarks

Sizes the window to its recommended optimal size.

---

## WinDefaultSize - Related Functions

- [WinInsertDefaultSize](#)
- 

## WinDefaultSize - Example Code

The following code illustrates how to size the window to its default size, or the optimum size of the dialog window.

```
#define INCL_SW
#include <os2me.h>

HWND hwndFrame;

hwndFrame = WinLoadSecondaryWindow (HWND_DESKTOP, /* Parent window */
                                     HWND_DESKTOP, /* Owner window */
                                     MyDlgProc,      /* Dialog procedure */
                                     NULL,           /* Module handle */
                                     ID_DIALOG,      /* Resource ID */
                                     NULL);          /* Create parameter */

WinInsertDefaultSize (hwndFrame, "~Default size"); /* Insert Menu Item */
WinDefaultSize (hwndFrame); /* Set window to its default size */
```

---

## WinDefaultSize - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## WinDestroySecondaryWindow

---

## WinDestroySecondaryWindow - Syntax



This function eliminates a secondary window.

```
#define INCL_SW
#include <os2.h>

HWND    hwnd; /* Secondary window to be eliminated. */
BOOL    rc;    /* Return codes. */

rc = WinDestroySecondaryWindow(hwnd);
```

-----

## WinDestroySecondaryWindow Parameter - hwnd

**hwnd** (**HWND**) - input  
Secondary window frame to be eliminated.

-----

## WinDestroySecondaryWindow Return Value - rc

**rc** (**BOOL**) - returns  
Return codes indicating success or failure:

TRUE	Success.
FALSE	Failure or not recognized.

-----

## WinDestroySecondaryWindow - Parameters

**hwnd** (**HWND**) - input  
Secondary window frame to be eliminated.

**rc** (**BOOL**) - returns  
Return codes indicating success or failure:

TRUE	Success.
FALSE	Failure or not recognized.

-----

## WinDestroySecondaryWindow - Related Functions

- [WinCreateSecondaryWindow](#)
- [WinLoadSecondaryWindow](#)

---

## WinDestroySecondaryWindow - Example Code

The following code illustrates how to destroy secondary windows.

```
#define INCL_SW
#include <os2me.h>

MRESULT EXPENTRY MyDlgProc (HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
    switch (msg) {
        case WM_CLOSE:
            /* Destroy a secondary window */
            WinDestroySecondaryWindow (hwndSecondaryFrame);
            return (TRUE);
            break;
    }

    return (WinDefSecondaryWindowProc (hwnd, msg, mp1, mp2));
}
```

---

## WinDestroySecondaryWindow - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## WinDismissSecondaryWindow

---

## WinDismissSecondaryWindow - Syntax

This function causes modal WinProcessSecondaryWindow or WinSecondaryWindow calls to return.

```
#define INCL_SW
#include <os2.h>
```

```
HWND    hwndDlg;    /* Secondary dialog window. */
ULONG    ulResult;  /* Result. */
BOOL     rc;         /* Return codes. */

rc = WinDismissSecondaryWindow(hwndDlg, ulResult);
```

---

## WinDismissSecondaryWindow Parameter - hwndDlg

**hwndDlg** ([HWND](#)) - input  
Secondary dialog window.

---

## WinDismissSecondaryWindow Parameter - ulResult

**ulResult** ([ULONG](#)) - input  
Result to be returned by [WinSecondaryWindow](#) or [WinProcessSecondaryWindow](#).

---

## WinDismissSecondaryWindow Return Value - rc

**rc** ([BOOL](#)) - returns  
Return codes indicating success or failure:

TRUE	Success.
FALSE	Failure or not recognized.

---

## WinDismissSecondaryWindow - Parameters

**hwndDlg** ([HWND](#)) - input  
Secondary dialog window.

**ulResult** ([ULONG](#)) - input  
Result to be returned by [WinSecondaryWindow](#) or [WinProcessSecondaryWindow](#).

**rc** ([BOOL](#)) - returns  
Return codes indicating success or failure:

TRUE	Success.
FALSE	Failure or not recognized.

---

## WinDismissSecondaryWindow - Remarks

Refer to WinDismissDlg in the *IBM OS/2 Presentation Manager Programming Reference*.

---

## WinDismissSecondaryWindow - Related Functions

- [WinCreateSecondaryWindow](#)
  - [WinDefSecondaryWindowProc](#)
  - [WinSecondaryWindow](#)
  - [WinLoadSecondaryWindow](#)
  - [WinProcessSecondaryWindow](#)
- 

## WinDismissSecondaryWindow - Example Code

The following code illustrates how to cause modal WinProcessSecondaryWindows or WinSecondaryWindow calls to return.

```
#define INCL_SW
#include <os2me.h>

case WM_COMMAND:
    switch (SHORT1FROMMP(mp1)) {
        case DID_CANCEL:
            /* Dismiss the window */
            WinDismissSecondaryWindow (hwnd, DID_CANCEL);
            break;
    }
```

---

## WinDismissSecondaryWindow - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## WinInsertDefaultSize

---

## WinInsertDefaultSize - Syntax

This function adds the default size function to the system menu of a secondary window.

```
#define INCL_SW
#include <os2.h>

HWND    hwnd;           /* Secondary window handle. */
PSZ     pszDefaultSize; /* Text for system menu. */
BOOL    rc;             /* Return codes. */

rc = WinInsertDefaultSize(hwnd, pszDefaultSize);
```

---

## WinInsertDefaultSize Parameter - hwnd

**hwnd** (**HWND**) - input  
Secondary window handle.

---

## WinInsertDefaultSize Parameter - pszDefaultSize

**pszDefaultSize** (**PSZ**) - input  
Text to be inserted into the system menu. Usually, this will read "Default Size".

---

## WinInsertDefaultSize Return Value - rc

**rc** (**BOOL**) - returns  
Return codes indicating success or failure:

TRUE	Success.
FALSE	Failure or not recognized.

---

## WinInsertDefaultSize - Parameters

**hwnd** (**HWND**) - input  
Secondary window handle.

**pszDefaultSize** ([PSZ](#)) - input

Text to be inserted into the system menu. Usually, this will read "Default Size".

**rc** ([BOOL](#)) - returns

Return codes indicating success or failure:

TRUE

Success.

FALSE

Failure or not recognized.

-----

## WinInsertDefaultSize - Remarks

The function adds **Default Size** to the system menu of a secondary window. Because a secondary window is a sizable dialog, its initial size is the optimal size for the dialog. After the window is sized, scroll bars appear when necessary. The default size function returns the window to its optimal size.

-----

## WinInsertDefaultSize - Related Functions

- [WinDefaultSize](#)

-----

## WinInsertDefaultSize - Example Code

The following code illustrates adding the default size function to the system menu of a secondary window.

```
#define INCL_SW
#include <os2me.h>

HWND hwndFrame;

hwndFrame = WinLoadSecondaryWindow (HWND_DESKTOP, /* Parent window */
                                     HWND_DESKTOP, /* Owner window */
                                     MyDlgProc,      /* Dialog procedure */
                                     NULL,           /* Module handle */
                                     ID_DIALOG,      /* Resource ID */
                                     NULL);          /* Create parameter */

WinInsertDefaultSize (hwndFrame, "~Default size"); /* Insert menu item */
WinDefaultSize (hwndFrame);                      /* Set window to its default size */
```

-----

## WinInsertDefaultSize - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## WinLoadSecondaryWindow

---

### WinLoadSecondaryWindow - Syntax

This function creates a secondary window from a dialog template in a resource. The secondary window is modeless.

```
#define INCL_SW
#include <os2.h>

HWND      hwndParent;    /* Parent window handle. */
HWND      hwndOwner;     /* Owner window handle. */
PFNWP     pfnDlgProc;    /* Secondary window procedure. */
HMODULE    hmod;         /* Module handle. */
ULONG     idDlg;         /* Resource ID of dialog template. */
PVOID     pCreateParams; /* Parameters passed to secondary window proc. */
HWND      hwnd;          /* Window handle. */

hwnd = WinLoadSecondaryWindow(hwndParent,
                               hwndOwner, pfnDlgProc, hmod, idDlg,
                               pCreateParams);
```

---

### WinLoadSecondaryWindow Parameter - hwndParent

**hwndParent** (**HWND**) - input  
Parent window handle of the created secondary frame window.

---

### WinLoadSecondaryWindow Parameter - hwndOwner

**hwndOwner** (**HWND**) - input  
Requested owner-window handle of the created secondary frame.

---

### WinLoadSecondaryWindow Parameter - pfnDlgProc

**pfnDlgProc** ([PFNWP](#)) - input

Secondary window procedure for the created dialog window. This is a dialog procedure, except that it calls [WinDefSecondaryWindowProc](#).

---

## WinLoadSecondaryWindow Parameter - hmod

**hmod** ([HMODULE](#)) - input

Module handle for resource module.

---

## WinLoadSecondaryWindow Parameter - idDlg

**idDlg** ([ULONG](#)) - input

Resource ID of dialog template in the resources of module *hmod*.

---

## WinLoadSecondaryWindow Parameter - pCreateParams

**pCreateParams** ([PVOID](#)) - input

Passed to the secondary window procedure in the WM\_INITDLG message.

---

## WinLoadSecondaryWindow Return Value - hwnd

**hwnd** ([HWND](#)) - returns

Returns the frame handle of the secondary window.

---

## WinLoadSecondaryWindow - Parameters

**hwndParent** ([HWND](#)) - input

Parent window handle of the created secondary frame window.

**hwndOwner** ([HWND](#)) - input

Requested owner-window handle of the created secondary frame.

**pfnDlgProc** ([PFNWP](#)) - input

Secondary window procedure for the created dialog window. This is a dialog procedure, except that it calls [WinDefSecondaryWindowProc](#).



**hmod** ([HMODULE](#)) - input  
Module handle for resource module.

**idDlg** ([ULONG](#)) - input  
Resource ID of dialog template in the resources of module *hmod*.

**pCreateParams** ([PVOID](#)) - input  
Passed to the secondary window procedure in the WM\_INITDLG message.

**hwnd** ([HWND](#)) - returns  
Returns the frame handle of the secondary window.

---

## WinLoadSecondaryWindow - Remarks

This call creates a standard frame window with a dialog window as the child of the FID\_CLIENT window. Refer to WinLoadDlg in the *IBM OS/2 Presentation Manager Programming Reference*.

---

## WinLoadSecondaryWindow - Related Functions

- [WinCreateSecondaryWindow](#)
  - [WinDefSecondaryWindowProc](#)
  - [WinDestroySecondaryWindow](#)
  - [WinDismissSecondaryWindow](#)
  - [WinSecondaryWindow](#)
  - [WinProcessSecondaryWindow](#)
- 

## WinLoadSecondaryWindow - Example Code

The following code illustrates creating a secondary window from a dialog template in a resource.

```
#define INCL_SW
#include <os2me.h>

HWND hwndFrame;

hwndFrame = WinLoadSecondaryWindow (HWND_DESKTOP, /* Parent window */
                                     HWND_DESKTOP, /* Owner window */
                                     MyDlgProc,      /* Dialog proc   */
                                     NULL,            /* Module handle */
                                     ID_DIALOG,      /* Resource ID   */
                                     NULL);          /* Create params */

WinShowWindow (hwndFrame, TRUE);
```

---

## WinLoadSecondaryWindow - Topics

Select an item:  
[Syntax](#)

[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## WinProcessSecondaryWindow

---

### WinProcessSecondaryWindow - Syntax

This function processes a modal secondary window by dispatching messages while the modal window is displayed.

```
#define INCL_SW
#include <os2.h>

HWND     hwndSW; /* Secondary window handle. */
ULONG     rc;     /* Result. */

rc = WinProcessSecondaryWindow(hwndSW);
```

---

### WinProcessSecondaryWindow Parameter - hwndSW

**hwndSW** (**HWND**) - input  
Secondary frame window handle to process.

---

### WinProcessSecondaryWindow Return Value - rc

**rc** (**ULONG**) - returns  
Returns the result passed to [WinDismissSecondaryWindow](#).

---

### WinProcessSecondaryWindow - Parameters

**hwndSW** (**HWND**) - input  
Secondary frame window handle to process.

rc ([ULONG](#)) - returns  
Returns the result passed to [WinDismissSecondaryWindow](#).

---

## WinProcessSecondaryWindow - Remarks

Refer to WinProcessDlg in the *IBM OS/2 Presentation Manager Programming Reference*.

---

## WinProcessSecondaryWindow - Related Functions

- [WinCreateSecondaryWindow](#)
  - [WinDefSecondaryWindowProc](#)
  - [WinDestroySecondaryWindow](#)
  - [WinDismissSecondaryWindow](#)
  - [WinSecondaryWindow](#)
- 

## WinProcessSecondaryWindow - Example Code

The following code illustrates processing a modal secondary window by dispatching messages while the modal window is displayed.

```
#define INCL_SW
#include <os2me.h>

HWND hwndFrame;

hwndFrame = WinLoadSecondaryWindow (HWND_DESKTOP, /* Parent window */
                                     HWND_DESKTOP, /* Owner window */
                                     MyDlgProc,     /* Dialog proc */
                                     NULL,           /* Module handle */
                                     ID_DIALOG,      /* Resource ID */
                                     NULL);          /* Create params */

WinProcessSecondaryWindow (hwndFrame);
```

---

## WinProcessSecondaryWindow - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

# WinQuerySecondaryHWND

---

## WinQuerySecondaryHWND - Syntax

This function returns a window handle to various windows created as a part of a secondary window.

```
#define INCL_SECONDARYWINDOW
#include <os2.h>

HWND hwnd; /* Secondary window handle. */
ULONG ulFlag; /* Flags. */
HWND hwnd; /* Requested window handle. */

hwnd = WinQuerySecondaryHWND(hwnd, ulFlag);
```

---

## WinQuerySecondaryHWND Parameter - hwnd

**hwnd** (**HWND**) - input  
Secondary window handle.

---

## WinQuerySecondaryHWND Parameter - ulFlag

**ulFlag** (**ULONG**) - input  
Flags.

QS_FRAME	Returns the HWND for the outer standard frame window.
QS_DIALOG	Returns the HWND for the inner dialog frame window.

---

## WinQuerySecondaryHWND Return Value - hwnd

**hwnd** (**HWND**) - returns  
Returns the requested window handle.

---

# WinQuerySecondaryHWND - Parameters

**hwnd** ([HWND](#)) - input  
Secondary window handle.

**ulFlag** ([ULONG](#)) - input  
Flags.

QS\_FRAME  
Returns the HWND for the outer standard frame window.

QS\_DIALOG  
Returns the HWND for the inner dialog frame window.

**hwnd** ([HWND](#)) - returns  
Returns the requested window handle.

---

## WinQuerySecondaryHWND - Remarks

This function returns the handle to the outer frame window or the inner dialog frame window of a secondary window, depending on the handle supplied as input.

Secondary windows utilize two frame windows: a standard frame and a dialog window to implement the sizing and scrolling features. The window handle returned from [WinLoadSecondaryWindow](#) is the handle to the standard frame. The window handle passed to the message procedure specified in [WinLoadSecondaryWindow](#) is the dialog window handle.

The standard frame window handle must be used when associating a help instance and to do WinSetWindowPos operations.

The dialog window handle should be used as the owner for message boxes and to access the controls on the dialog with WinWindowFromID.

To modify the title bar or the system menu, the application must specify the standard frame window and not the dialog window.

---

## WinQuerySecondaryHWND - Related Functions

- [WinCreateSecondaryWindow](#)
- [WinDefSecondaryWindowProc](#)
- [WinDestroySecondaryWindow](#)
- [WinDismissSecondaryWindow](#)
- [WinLoadSecondaryWindow](#)
- [WinSecondaryWindow](#)

---

## WinQuerySecondaryHWND - Example Code

The following code illustrates returning a window handle to various windows created as a part of a secondary window.

```
#define INCL_SECONDARYWINDOW
#include <sw.h>

HWND hwndFrame, hwndDialog;
```

```

hwndFrame = WinLoadSecondaryWindow (HWND_DESKTOP, /* Parent window */
                                     HWND_DESKTOP, /* Owner window */
                                     MyDlgProc,    /* Dialog proc */
                                     NULL,         /* Module handle */
                                     ID_DIALOG,    /* Resource ID */
                                     NULL);        /* Create params */
hwndDialog = WinQuerySecondaryHWND( hwnd, QS_DIALOG );

```

-----

## WinQuerySecondaryHWND - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

-----

## WinSecondaryMessageBox

-----

## WinSecondaryMessageBox - Syntax

This function creates a modal window, similar to WinMessageBox, that can be used to display error messages and ask questions.

```

#define INCL_SW
#include <os2.h>

HWND      hwndParent; /* Parent window handle. */
HWND      hwndOwner;  /* Owner window handle. */
PSZ       pszText;    /* Message text. */
PSZ       pszCaption; /* Title of secondary message box window. */
ULONG     idWindow;   /* ID of window. */
PSMBINFO  psmbinfo;   /* Information block. */
ULONG     rc;         /* Results. */

rc = WinSecondaryMessageBox(hwndParent, hwndOwner,
                           pszText, pszCaption, idWindow, psmbinfo);

```

-----

## WinSecondaryMessageBox Parameter - hwndParent

**hwndParent** ([HWND](#)) - input

The parent window handle of the secondary window to be created.

---

## WinSecondaryMessageBox Parameter - hwndOwner

**hwndOwner** ([HWND](#)) - input

The owner-window handle of the secondary window to be created.

---

## WinSecondaryMessageBox Parameter - pszText

**pszText** ([PSZ](#)) - input

The text of the message to be displayed.

---

## WinSecondaryMessageBox Parameter - pszCaption

**pszCaption** ([PSZ](#)) - input

The title of the secondary message box window.

---

## WinSecondaryMessageBox Parameter - idWindow

**idWindow** ([ULONG](#)) - input

The identifier of the secondary message box window.

---

## WinSecondaryMessageBox Parameter - psmbinfo

**psmbinfo** ([PSMBINFO](#)) - input

This is an information block that defines which buttons must be included in the window.

---

## WinSecondaryMessageBox Return Value - rc

**rc** ([ULONG](#)) - returns

Returns the result passed to [WinDismissSecondaryWindow](#); this is the ID of the button that was clicked.

---

## WinSecondaryMessageBox - Parameters

**hwndParent** ([HWND](#)) - input

The parent window handle of the secondary window to be created.

**hwndOwner** ([HWND](#)) - input

The owner-window handle of the secondary window to be created.

**pszText** ([PSZ](#)) - input

The text of the message to be displayed.

**pszCaption** ([PSZ](#)) - input

The title of the secondary message box window.

**idWindow** ([ULONG](#)) - input

The identifier of the secondary message box window.

**psmbinfo** ([PSMBINFO](#)) - input

This is an information block that defines which buttons must be included in the window.

**rc** ([ULONG](#)) - returns

Returns the result passed to [WinDismissSecondaryWindow](#); this is the ID of the button that was clicked.

---

## WinSecondaryMessageBox - Remarks

Refer to WinMessageBox in the *IBM OS/2 Presentation Manager Programming Reference*.

---

## WinSecondaryMessageBox - Related Functions

- [WinCreateSecondaryWindow](#)
  - [WinDefSecondaryWindowProc](#)
  - [WinDestroySecondaryWindow](#)
  - [WinDismissSecondaryWindow](#)
  - [WinLoadSecondaryWindow](#)
  - [WinSecondaryWindow](#)
- 

## WinSecondaryMessageBox - Example Code

The following code illustrates creating a modal window, similar to WinMessageBox, that can be used to display error messages and ask questions.

```
#define INCL_SW
#include <os2me.h>
```



```

SMBD Smbd[4] = { { "~Save",      ID_SAVE,BS_DEFAULT},
                  { "~Discard",  ID_DISCARD,0},
                  { "Cancel",    ID_CANCEL,  0},
                  { "Help",      ID_HELP,    BS_HELP}}};

SMBINFO SmbInfo;
ULONG   Result;

SmbInfo.hIcon = NULL;
SmbInfo.cButtons = 4;
SmbInfo.hwndNotify = NULL;
SmbInfo.flStyle = MB_QUERY;
SmbInfo.psmdb = Smbd;

Result = WinSecondaryMessageBox (HWND_DESKTOP,
                                HWND_DESKTOP,
                                "Changes have not been saved.",
                                "Title",
                                ID_SAVEPROMPT,
                                &SmbInfo);

```

---

## WinSecondaryMessageBox - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## WinSecondaryWindow

---

## WinSecondaryWindow - Syntax

This function loads and processes a modal secondary window and returns the result value established by [WinDismissSecondaryWindow](#).

```

#define INCL_SW
#include <os2.h>

HWND      hwndParent;      /* Parent window handle. */
HWND      hwndOwner;       /* Owner window handle. */
PFNWP     pfndlgProc;      /* Secondary window procedure. */
HMODULE    hmod;           /* Module handle. */
ULONG     idDlg;           /* Resource ID of dialog template. */
PVOID     pCreateParams;   /* Parameters passed to secondary window proc. */
ULONG     rc;              /* Results. */

rc = WinSecondaryWindow(hwndParent, hwndOwner,
                        pfndlgProc, hmod, idDlg, pCreateParams);

```

---

## WinSecondaryWindow Parameter - hwndParent

**hwndParent** ([HWND](#)) - input  
The parent window handle of the [WinCreateSecondaryWindow](#) frame.

---

## WinSecondaryWindow Parameter - hwndOwner

**hwndOwner** ([HWND](#)) - input  
The owner-window handle of the [WinCreateSecondaryWindow](#) frame.

---

## WinSecondaryWindow Parameter - pfnDlgProc

**pfnDlgProc** ([PFNWP](#)) - input  
The secondary window procedure for [CreateDialogWindow](#). This is a dialog procedure, except that it calls [WinDefSecondaryWindowProc](#).

---

## WinSecondaryWindow Parameter - hmod

**hmod** ([HMODULE](#)) - input  
Module handle for resource module.

---

## WinSecondaryWindow Parameter - idDlg

**idDlg** ([ULONG](#)) - input  
Resource ID of dialog template in the resources of module *hmod*.

---

## WinSecondaryWindow Parameter - pCreateParams

**pCreateParams** ([PVOID](#)) - input  
This is passed to the secondary window procedure in the [WM\\_INITDLG](#) message.

---

## WinSecondaryWindow Return Value - rc

**rc** ([ULONG](#)) - returns  
Return the result passed to [WinDismissSecondaryWindow](#).

---

## WinSecondaryWindow - Parameters

**hwndParent** ([HWND](#)) - input  
The parent window handle of the [WinCreateSecondaryWindow](#) frame.

**hwndOwner** ([HWND](#)) - input  
The owner-window handle of the [WinCreateSecondaryWindow](#) frame.

**pfnDlgProc** ([PFNWP](#)) - input  
The secondary window procedure for [CreateDialogWindow](#). This is a dialog procedure, except that it calls [WinDefSecondaryWindowProc](#).

**hmod** ([HMODULE](#)) - input  
Module handle for resource module.

**idDlg** ([ULONG](#)) - input  
Resource ID of dialog template in the resources of module *hmod*.

**pCreateParams** ([PVOID](#)) - input  
This is passed to the secondary window procedure in the WM\_INITDLG message.

**rc** ([ULONG](#)) - returns  
Return the result passed to [WinDismissSecondaryWindow](#).

---

## WinSecondaryWindow - Remarks

This call creates a standard frame window with a dialog window as the child of FID\_CLIENT. Refer to WinDlgBox in the *IBM OS/2 Presentation Manager Programming Reference*.

---

## WinSecondaryWindow - Related Functions

- [WinCreateSecondaryWindow](#)
  - [WinDefSecondaryWindowProc](#)
  - [WinDestroySecondaryWindow](#)
  - [WinDismissSecondaryWindow](#)
  - [WinLoadSecondaryWindow](#)
-

# WinSecondaryWindow - Example Code

The following code illustrates how to load and process a modal secondary window and returns the result value established by [WinDismissSecondaryWindow](#).

```
#define INCL_SW
#include <os2me.h>

ULONG Result;

Result = WinSecondaryWindow (HWND_DESKTOP,
                             HWND_DESKTOP,
                             MyDlgProc,
                             NULL,
                             ID_DIALOG,
                             NULL);
```

## WinSecondaryWindow - Topics

- Select an item:
- [Syntax](#)
  - [Parameters](#)
  - [Returns](#)
  - [Remarks](#)
  - [Example Code](#)
  - [Related Functions](#)
  - [Glossary](#)

## Secondary Window Data Structures

This section describes the secondary window data structures. The following table describes the secondary window data structures.

Data Type	Description
<a href="#">SMBD</a>	Defines the button style, text, and ID for each button to be included in a secondary message box.
<a href="#">SMBINFO</a>	Defines the icon, number of buttons, and window style of the secondary message box window.

## MMIO Functions

The multimedia input/output (MMIO) file services are an extension of the base OS/2 file services. Designed to be simple, fast, and flexible, the MMIO services-functions, messages, and data structures-enable an application to access and manipulate multimedia data files. These files contain a variety of media elements (images, graphics, digital audio, and video) that are in different file formats; for example, RIFF,

AVC, and M-Motion.

MMIO services provide a consistent programming interface so that an application can refer to these files, read and write data to the files, and query the contents of the files, without having to know anything about the specific format of the files. MMIO services strengthen a program's portability, as well as data compatibility, by insulating the application from the underlying file formats.

**Note:** As a general comment, all fields in MMIO services data structures that are not used in a given function must be initialized to NULL. Flags that are not used, and bits not used within flags, must be set to 0.

The following table describes the MMIO functions.

Function	Description
<code>mmioAdvance</code>	Fills and empties the contents of an I/O buffer of a file set up for direct I/O buffer access.
<code>mmioAscend</code>	Ascends out of a chunk in a RIFF file that was descended into by <code>mmioDescend</code> or created by <code>mmioCreateChunk</code> .
<code>mmioCFAddElement</code>	Adds an element to the CGRP chunk of an open RIFF compound file.
<code>mmioCFAddEntry</code>	Adds an entry to the CTOC chunk of an open RIFF compound file.
<code>mmioCFChangeEntry</code>	Changes a CTOC entry in an open RIFF compound file.
<code>mmioCFClose</code>	Closes a RIFF compound file that was opened by <code>mmioCFOpen</code> .
<code>mmioCFCompact</code>	Compacts the elements of a RIFF compound file.
<code>mmioCFCopy</code>	Copies the CTOC and CGRP chunks from an open RIFF compound file to another RIFF compound file.
<code>mmioCFDeleteEntry</code>	Deletes a CTOC entry in an open RIFF compound file.
<code>mmioCFFindEntry</code>	Finds a CTOC entry in an open RIFF compound file.
<code>mmioCFGetInfo</code>	Retrieves the CTOC header of an open RIFF compound file.
<code>mmioCFOpen</code>	Opens a RIFF compound file by name.
<code>mmioCFSetInfo</code>	Modifies information that is stored in the CTOC header of an open RIFF compound file.
<code>mmioClose</code>	Closes a file opened by <code>mmioOpen</code> .
<code>mmioCreateChunk</code>	Creates a chunk in a RIFF file that was opened by <code>mmioOpen</code> .
<code>mmioDescend</code>	Descends into a RIFF file chunk beginning at the current file position, or searches for a specified chunk.
<code>mmioDetermineSSIOProc</code>	Determines the storage system of the media data object.
<code>mmioFindElement</code>	Enumerates the entries of a compound file.
<code>mmioFlush</code>	Forces the contents of an I/O buffer to be written to disk.
<code>mmioFOURCC</code>	Converts four characters to a four-character code (FOURCC).

<code>mmioGetFormatName</code>	Provides the descriptive name of the format supported by the IOProc.
<code>mmioGetFormats</code>	Provides a list of all format I/O procedures available for use.
<code>mmioGetHeader</code>	Obtains media-specific information about data in a file such as the media type, media structure, and the size of the media structure.
<code>mmioGetInfo</code>	Retrieves information from the file I/O buffer to a file opened for buffered I/O.
<code>mmioGetLastError</code>	Returns the last error condition stored in <code>ulErrorRet</code> that might contain additional information for the analysis of the last error routine.
<code>mmioIdentifyFile</code>	Determines (if possible) the format of a file by either using the file name or querying currently installed I/O procedures to see which IOProc can understand and process the specified file.
<code>mmioIdentifyStorageSystem</code>	Identifies the storage system that contains the media data object.
<code>mmioIniFileCODEC</code>	Modifies the initialization file (MMPMMMIO.INI) for MMIO services. It adds, replaces, removes, or finds a CODEC procedure in the MMPMMMIO.INI file.
<code>mmioIniFileHandler</code>	Adds, replaces, removes, or finds an I/O procedure in the initialization file (MMPMMMIO.INI).
<code>mmioInstallIOProc</code>	Installs an I/O procedure in the MMIO IOProc table, removes an IOProc from the table, or finds a procedure when given its FOURCC identifier.
<code>mmioLoadCODECProc</code>	Loads the CODEC Proc and returns the entry point.
<code>mmioOpen</code>	Opens a file and returns an MMIO handle.
<code>mmioQueryCODECName</code>	Returns the CODEC Proc name.
<code>mmioQueryCODECNameLength</code>	Returns the length of the CODEC Proc name.
<code>mmioQueryFormatCount</code>	Provides the number of IOProcs that match the requested format.
<code>mmioQueryHeaderLength</code>	Determines the size of the header for a specified file.
<code>mmioQueryIOProcModuleHandle</code>	Provides the module handle of an IOProc's DLL. This handle must be used to retrieve resources from the DLL. This function provides the handle of the DLL only if it was loaded by MMIO from the MMPMMMIO.INI file.
<code>mmioRead</code>	Reads a specified number of bytes from a file opened by <code>mmioOpen</code> .
<code>mmioRemoveElement</code>	Removes the specified element in a compound file.
<code>mmioSeek</code>	Changes the current position for reading, writing, or both, in a file opened by <code>mmioOpen</code> .
<code>mmioSendMessage</code>	Sends a message to the I/O procedure

	associated with a file that was opened with <code>mmioOpen</code> .
<code>mmioSet</code>	Sets or queries extended file information.
<code>mmioSetBuffer</code>	Enables or disables buffered I/O, or changes the buffer or buffer size, for a file that was opened using <code>mmioOpen</code> .
<code>mmioSetHeader</code>	Sets media-specific information for data to be written to a file.
<code>mmioSetInfo</code>	Changes information on a file I/O buffer of a file opened for buffered I/O.
<code>mmioStringToFOURCC</code>	Converts a null-terminated string to a four-character code (FOURCC).
<code>mmioWrite</code>	Writes to a file that was opened using <code>mmioOpen</code> .

-----

## mmioAdvance

-----

## mmioAdvance - Syntax

This function fills and empties the contents of an I/O buffer of a file set up for direct I/O buffer manipulation by `mmioGetInfo`.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO      hmmio;      /* Open file handle. */
PMMIOINFO  pmmioinfo; /* Pointer to MMIOINFO. */
USHORT     usFlags;    /* Flags. */
USHORT     rc;         /* Return codes. */

rc = mmioAdvance(hmmio, pmmioinfo, usFlags);
```

-----

## mmioAdvance Parameter - hmmio

**hmmio** (**HMMIO**) - input  
The open file handle returned by `mmioOpen`.

-----

## mmioAdvance Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#)) - input

A pointer to the [MMIOINFO](#) data structure that was filled in by [mmioGetInfo](#).

---

## mmioAdvance Parameter - usFlags

**usFlags** ([USHORT](#)) - input

Specifies options for the operation. Contains one or more of the following flags:

MMIO\_READ

The buffer is refilled from the file. MMIO\_READ is used when the caller has finished reading data from the I/O buffer and wants the buffer to be refilled (if possible).

MMIO\_WRITE

The buffer is written to the file and not refilled from the file. MMIO\_WRITE is used when the caller has written to the end of the buffer and needs the buffer to be emptied (or expanded, in the case of a memory file).

---

## mmioAdvance Return Value - rc

**rc** ([USHORT](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_UNBUFFERED

The specified file is not opened for buffered I/O.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_READ\_ONLY\_FILE

A write-advance operation was requested for a read-only file.

MMIOERR\_WRITE\_ONLY\_FILE

A read-advance operation was requested for a file opened as write only.

MMIOERR\_WRITE\_FAILED

A write-advance operation failed.

MMIOERR\_READ\_FAILED

A read-advance operation failed.

MMIOERR\_SEEK\_FAILED

A seek operation prior to a write- or read-advance operation failed.

MMIOERR\_NO\_FLUSH\_NEEDED

A write-advance operation was requested for the buffer, but the operation was not required.

MMIOERR\_OUTOFMEMORY

An advance operation requires a buffer.

MMIOERR\_CANNOTEXPAND

Unable to expand a MEM file for an advance request.



MMIOERR\_FREE\_FAILED

Unable to free a buffer after expanding a MEM file.

-----

## mmioAdvance - Parameters

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

**pmmioinfo** ([PMMIOINFO](#)) - input

A pointer to the [MMIOINFO](#) data structure that was filled in by [mmioGetInfo](#).

**usFlags** ([USHORT](#)) - input

Specifies options for the operation. Contains one or more of the following flags:

MMIO\_READ

The buffer is refilled from the file. MMIO\_READ is used when the caller has finished reading data from the I/O buffer and wants the buffer to be refilled (if possible).

MMIO\_WRITE

The buffer is written to the file and not refilled from the file. MMIO\_WRITE is used when the caller has written to the end of the buffer and needs the buffer to be emptied (or expanded, in the case of a memory file).

**rc** ([USHORT](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_UNBUFFERED

The specified file is not opened for buffered I/O.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_READ\_ONLY\_FILE

A write-advance operation was requested for a read-only file.

MMIOERR\_WRITE\_ONLY\_FILE

A read-advance operation was requested for a file opened as write only.

MMIOERR\_WRITE\_FAILED

A write-advance operation failed.

MMIOERR\_READ\_FAILED

A read-advance operation failed.

MMIOERR\_SEEK\_FAILED

A seek operation prior to a write- or read-advance operation failed.

MMIOERR\_NO\_FLUSH\_NEEDED

A write-advance operation was requested for the buffer, but the operation was not required.

MMIOERR\_OUTOFMEMORY

An advance operation requires a buffer.

MMIOERR\_CANNOTEXPAND

Unable to expand a MEM file for an advance request.

MMIOERR\_FREE\_FAILED

Unable to free a buffer after expanding a MEM file.

---

## mmioAdvance - Remarks

The mmioAdvance function does not change the current file position of the file represented by the *hmmio* parameter, that is, *pchNext* of the [MMIOINFO](#) structure passed in the *pmmioinfo* parameter will correspond to the same data position before and after mmioAdvance is called, hence pointing to the same piece of data that is now located at the beginning of the buffer. mmioAdvance simply makes available as much buffer space as possible for doing direct buffer reading or writing.

When mmioAdvance returns from a call where the MMIO\_READ flag was specified, there will be at least *n* bytes of data available to be read in the I/O buffer (that is, *n* bytes between *pchNext* and *pchEndRead*), where *n* is the lesser of the I/O buffer size and the number of remaining bytes of data.

When mmioAdvance returns from a call where the MMIO\_WRITE flag was specified, at least *n* bytes of free space in the I/O buffer (that is, *n* bytes between *pchNext* and *pchEndWrite*) where *n* is specified in the *availInfo[0]* field of the [MMIOINFO](#) structure passed on an [mmioOpen](#) called if the file is a memory file, or the size of the I/O buffer if the file is not a memory file.

If the file is opened for reading, the I/O buffer is filled from the disk. If the file is opened for writing and the MMIO\_DIRTY flag is set in the *ulFlags* field of the [MMIOINFO](#) structure, the buffer is written to disk. The *pchNext*, *pchEndRead*, and *pchEndWrite* fields of the [MMIOINFO](#) structure are updated to reflect the new state of the I/O buffer.

If the file was opened for both reading and writing, and the I/O buffer was written to, the contents of the I/O buffer are written to disk before the next buffer is read.

If you have written to the I/O buffer, you must set the MMIO\_DIRTY flag of the *ulFlags* field of the [MMIOINFO](#) structure before calling mmioAdvance. Otherwise, the buffer will not be written to disk.

The *pchNext* field must also be updated to reflect the data written in the I/O buffer. The I/O buffer will be written up to (but not including) the position indicated by the *pchNext* field of [MMIOINFO](#).

---

## mmioAdvance - Related Functions

- [mmioGetInfo](#)
  - [mmioSetInfo](#)
- 

## mmioAdvance - Example Code

The following code illustrates how to advance the I/O buffer.

```
HMMIO hmmiol;
MMIOINFO mmioinfo;
USHORT usFlags;
USHORT rc;
...

rc = mmioAdvance(hmmiol, &mmioinfo, usFlags);
if (rc)
    /* error */
else
    ...
```

---

## mmioAdvance - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

---

## mmioAscend

---

### mmioAscend - Syntax

This function ascends out of a chunk in a RIFF file that was descended into by [mmioDescend](#) or created by [mmioCreateChunk](#).

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO      hmmio;    /* Open file handle. */
PMMCKINFO  pckinfo;  /* Pointer to MMCKINFO. */
USHORT     usFlags;   /* Reserved. */
USHORT     rc;        /* Return codes. */

rc = mmioAscend(hmmio, pckinfo, usFlags);
```

---

### mmioAscend Parameter - hmmio

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

---

### mmioAscend Parameter - pckinfo

**pckinfo** ([PMMCKINFO](#)) - input

A pointer to the [MMCKINFO](#) structure that was filled by [mmioDescend](#) or [mmioCreateChunk](#).

---

### mmioAscend Parameter - usFlags

**usFlags** ([USHORT](#)) - input  
Reserved for future use and must be set to zero.

---

## mmioAscend Return Value - rc

**rc** ([USHORT](#)) - returns  
Return codes indicating success or type of failure:

- MMIO\_SUCCESS**  
If the function succeeds, 0 is returned.
- MMIOERR\_INVALID\_HANDLE**  
The handle passed was not valid.
- MMIOERR\_INVALID\_PARAMETER**  
The parameter passed was not correct.
- MMIOERR\_CANNOTWRITE**  
The I/O buffer needs to be written to disk but disk space is lacking.

---

## mmioAscend - Parameters

**hmmio** ([HMMIO](#)) - input  
The open file handle returned by [mmioOpen](#).

**pckinfo** ([PMMCKINFO](#)) - input  
A pointer to the [MMCKINFO](#) structure that was filled by [mmioDescend](#) or [mmioCreateChunk](#).

**usFlags** ([USHORT](#)) - input  
Reserved for future use and must be set to zero.

**rc** ([USHORT](#)) - returns  
Return codes indicating success or type of failure:

- MMIO\_SUCCESS**  
If the function succeeds, 0 is returned.
- MMIOERR\_INVALID\_HANDLE**  
The handle passed was not valid.
- MMIOERR\_INVALID\_PARAMETER**  
The parameter passed was not correct.
- MMIOERR\_CANNOTWRITE**  
The I/O buffer needs to be written to disk but disk space is lacking.

---

## mmioAscend - Remarks

If the chunk was descended into using [mmioDescend](#), then mmioAscend seeks to the location following the end of the chunk (past the extra

pad byte, if any). If the chunk was created and descended into using [mmioCreateChunk](#) or if the MMIO\_DIRTY flag in the *ulFlags* field (of the [MMCKINFO](#) structure passed in the *pckinfo* parameter) is set, then the current file position is assumed to mark the end of the data portion of the chunk. If the chunk size is not the same as the value that was stored in the *ckSize* field of [MMCKINFO](#) before [mmioCreateChunk](#) or [mmioDescend](#) was called, then mmioAscend seeks back and corrects the chunk size in the chunk header before ascending from the chunk. Also, if the chunk size is odd, then mmioAscend writes a NULL pad byte at the end of the chunk.

---

## mmioAscend - Related Functions

- [mmioCreateChunk](#)
- [mmioDescend](#)
- [mmioFOURCC](#)
- [mmioStringToFOURCC](#)

---

## mmioAscend - Example Code

The following code illustrates how to move the file position.

```
HMMIO hmmiol;
MMCKINFO mmckinfo;
USHORT usFlags;
USHORT rc;
...

memset( &mmckinfo, '\0', sizeof(MMCKINFO) );
rc = mmioAscend(hmmiol, &mmckinfo, usFlags);
if (rc)
    /* error */
else
    ...
```

---

## mmioAscend - Topics

- Select an item:
- [Syntax](#)
  - [Parameters](#)
  - [Returns](#)
  - [Remarks](#)
  - [Example Code](#)
  - [Related Functions](#)
  - [Glossary](#)

---

## mmioCFAddElement

---

## mmioCFAddElement - Syntax

This function adds an element to the compound-file resource-group (CGRP) chunk of an open RIFF compound file.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMCF    hmmcf;          /* RIFF file handle. */
PSZ      pszElementName; /* Pointer to the name of the element. */
FOURCC   fccType;        /* Four-character code. */
PCHAR    pchBuffer;      /* Buffer pointer. */
LONG     cchBytes;        /* Length of buffer. */
ULONG    ulFlags;         /* Flags. */
ULONG    rc;              /* Return codes. */

rc = mmioCFAddElement(hmmcf, pszElementName,
                     fccType, pchBuffer, cchBytes, ulFlags);
```

---

## mmioCFAddElement Parameter - hmmcf

**hmmcf** ([HMMCF](#)) - input  
A RIFF compound-file handle returned by [mmioCFOpen](#).

---

## mmioCFAddElement Parameter - pszElementName

**pszElementName** ([PSZ](#)) - input  
A pointer to the name of the element that is to be added to the compound file resource group (CGRP). The symbols + and | are not valid with an element name. Element names follow the same naming conventions as those for the DOS operating system.

---

## mmioCFAddElement Parameter - fccType

**fccType** ([FOURCC](#)) - input  
The four-character code (FOURCC) of the element.

---

## mmioCFAddElement Parameter - pchBuffer

**pchBuffer** ([PCHAR](#)) - input  
A pointer to the caller-supplied buffer containing the element data.

---

## mmioCFAddElement Parameter - cchBytes

**cchBytes** ([LONG](#)) - input  
Length of caller-supplied buffer.

---

## mmioCFAddElement Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input  
Specifies options for the operation. Contains 0 or the following flag:

**MMIO\_CF\_ENTRY\_EXISTS**  
Set only if the compound-file table of contents (CTOC) entry for this element already exists.

---

## mmioCFAddElement Return Value - rc

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure:

**MMIO\_CF\_SUCCESS**  
If the function succeeds, 0 is returned.

**MMIOERR\_INVALID\_HANDLE**  
The handle passed was not valid.

**MMIOERR\_INVALID\_PARAMETER**  
For this function a null *pszElementName*, *pchBuffer*, or *cchBytes* is invalid.

**MMIOERR\_READ\_ONLY\_FILE**  
The RIFF compound file is opened as read-only.

**MMIO\_CF\_FAILURE**  
The function failed. A call to [mmioGetLastError](#) may return:

**MMIOERR\_INTERNAL\_SYSTEM**  
The operation failed due to an internal system error.

---

## mmioCFAddElement - Parameters

**hmmcf** ([HMMCF](#)) - input  
A RIFF compound-file handle returned by [mmioCFOpen](#).

**pszElementName** ([PSZ](#)) - input  
A pointer to the name of the element that is to be added to the compound file resource group (CGRP). The symbols + and | are not valid with an element name. Element names follow the same naming conventions as those for the DOS operating system.

**fccType** ([FOURCC](#)) - input  
The four-character code (FOURCC) of the element.

**pchBuffer** ([PCHAR](#)) - input  
A pointer to the caller-supplied buffer containing the element data.

**cchBytes** ([LONG](#)) - input  
Length of caller-supplied buffer.

**ulFlags** ([ULONG](#)) - input  
Specifies options for the operation. Contains 0 or the following flag:

MMIO\_CF\_ENTRY\_EXISTS  
Set only if the compound-file table of contents (CTOC) entry for this element already exists.

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure:

MMIO\_CF\_SUCCESS  
If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE  
The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER  
For this function a null *pszElementName*, *pchBuffer*, or *cchBytes* is invalid.

MMIOERR\_READ\_ONLY\_FILE  
The RIFF compound file is opened as read-only.

MMIO\_CF\_FAILURE  
The function failed. A call to [mmioGetLastError](#) may return:

MMIOERR\_INTERNAL\_SYSTEM  
The operation failed due to an internal system error.

-----

## mmioCFAddElement - Remarks

The compound-file table of contents (CTOC) entry for the element does not have to exist prior to the call. [mmioCFAddElement](#) adds the entry if it does not exist. The [mmioCFAddElement](#) function writes the element to the end of the compound-file resource-group (CGRP) chunk. The user's buffer contains the element data.

This function is used when the element exists but is not contained in the RIFF compound file. If the element does not exist, use [mmioOpen](#) with the MMIO\_CREATE flag to add a newly created element to the RIFF compound file. In that case the file name used with [mmioOpen](#) must contain the compound file and element name (that is, *aaa.bnd+element*).

The user is not required to use [mmioCFAddElement](#) to add an element to a RIFF compound file. However, one would need to replicate the following function that [mmioCFAddElement](#) provides.

- Seek to the start of the RIFF compound file.
- Descend to the BND chunk.
- Descend to the CGRP chunk.
- Seek to the end of the CGRP.
- [mmioWrite](#) to write all of the data.
- Ascend the CGRP to correct the size.
- Ascend the BND to correct the size.
- If the CTOC already exists, call [mmioCFChangeEntry](#) to update the data offset and size of the element.
- If not, call [mmioCFAddEntry](#) to add the CTOC entry.

-----

## mmioCFAddElement - Related Functions



- [mmioCFCopy](#)

---

## mmioCFAddElement - Example Code

The following code illustrates how to add an element.

```
HMMCF hmmcf1;
CHAR *pFileName;
FOURCC fcctype;
CHAR *pchBuffer;
USHORT cchBuffer;
ULONG ulFlags;
ULONG rc;
...

strcpy( pFileName, "myelement.foo" );
fcctype = FOURCC_FOO;
rc = mmioCFAddElement(hmmcf1,
                      pFileName,
                      fcctype,
                      pchBuffer,
                      cchBuffer,
                      ulFlags);

if (rc)
    /* error */
else
    ...
```

---

## mmioCFAddElement - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioCFAddEntry

---

## mmioCFAddEntry - Syntax

This function adds an entry to the compound-file table of contents (CTOC) chunk of an open RIFF compound-file. Duplicate entries are not

allowed.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMCF      hmmcf;          /* RIFF file handle. */
PMMCTOCENTRY pmmctocentry; /* Pointer to MMCTOCENTRY. */
ULONG      ulFlags;        /* Flags. */
ULONG      rc;             /* Return codes. */

rc = mmioCFAddEntry(hmmcf, pmmctocentry, ulFlags);
```

---

## mmioCFAddEntry Parameter - hmmcf

**hmmcf** ([HMMCF](#)) - input  
A RIFF compound-file handle returned by [mmioCFOpen](#).

---

## mmioCFAddEntry Parameter - pmmctocentry

**pmmctocentry** ([PMMCTOCENTRY](#)) - input  
A pointer to a user-supplied CTOC structure containing the CTOC data. This structure is variable in size and the user must ensure enough memory has been allocated for it.

---

## mmioCFAddEntry Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input  
(Flags currently not used.)

---

## mmioCFAddEntry Return Value - rc

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure:

- MMIO\_CF\_SUCCESS**  
If the function succeeds, 0 is returned.
- MMIOERR\_INVALID\_HANDLE**  
The handle passed was not valid.
- MMIOERR\_INVALID\_PARAMETER**  
The parameter passed was not correct. For this function, a *pmmctocentry* NULL is invalid.

MMIOERR\_READ\_ONLY\_FILE

The RIFF compound-file is opened as read-only.

MMIO\_CF\_FAILURE

The function failed. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_CF\_DUPLICATE\_SEEN

The requested entry already exists. This means that the element name is already defined in another CTOC entry.

MMIOERR\_NO\_CORE

Not enough memory is available.

MMIOERR\_INTERNAL\_SYSTEM

The operation failed due to an internal system error.

-----

## mmioCFAddEntry - Parameters

**hmmcf** ([HMMCF](#)) - input

A RIFF compound-file handle returned by [mmioCFOpen](#).

**pmmctocentry** ([PMMCTOCENTRY](#)) - input

A pointer to a user-supplied CTOC structure containing the CTOC data. This structure is variable in size and the user must ensure enough memory has been allocated for it.

**ulFlags** ([ULONG](#)) - input

(Flags currently not used.)

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_CF\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

The parameter passed was not correct. For this function, a *pmmctocentry* NULL is invalid.

MMIOERR\_READ\_ONLY\_FILE

The RIFF compound-file is opened as read-only.

MMIO\_CF\_FAILURE

The function failed. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_CF\_DUPLICATE\_SEEN

The requested entry already exists. This means that the element name is already defined in another CTOC entry.

MMIOERR\_NO\_CORE

Not enough memory is available.

MMIOERR\_INTERNAL\_SYSTEM

The operation failed due to an internal system error.

-----

## mmioCFAddEntry - Remarks

The identifier for the entry is the element name, which is appended to the end of the [MMCTOCENTRY](#) structure passed in the

*pmmctocentry* parameter. It is not case-sensitive. If mmioCFAddEntry expands the current number of entries past the number currently allocated, when a [mmioCFClose](#) is called, the CTOC gets written following the CGRP chunk in the file. The mmioCFAddEntry function operates only on the CTOC in memory and does not do any expansion of the file itself.

---

## mmioCFAddEntry - Related Functions

- [mmioCFChangeEntry](#)
  - [mmioCFFindEntry](#)
  - [mmioCFDeleteEntry](#)
- 

## mmioCFAddEntry - Example Code

The following code illustrates how to add an entry.

```
HMMCF hmmcfl;
MMCTOCENTRY mmctocentry;
ULONG ulFlags;
ULONG rc;
...

ulFlags = 0;
rc = mmioCFAddEntry(hmmcfl,
                    &mmctocentry,
                    ulFlags);

if (rc)
    /* error */
else
    ...
```

---

## mmioCFAddEntry - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioCFChangeEntry

---

## mmioCFChangeEntry - Syntax

This function modifies a compound-file table of contents (CTOC) entry in an open RIFF compound file.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMCF      hmmcf;      /* RIFF file handle. */
PMMCTOCENTRY pmmctocentry; /* Pointer to MMCTOCENTRY. */
ULONG      ulFlags;     /* Flags. */
ULONG      rc;          /* Return codes. */

rc = mmioCFChangeEntry(hmmcf, pmmctocentry,
                      ulFlags);
```

---

## mmioCFChangeEntry Parameter - hmmcf

**hmmcf** ([HMMCF](#)) - input  
A RIFF compound-file handle returned by [mmioCFOpen](#).

---

## mmioCFChangeEntry Parameter - pmmctocentry

**pmmctocentry** ([PMMCTOCENTRY](#)) - input  
A pointer to the [MMCTOCENTRY](#) structure containing the modified CTOC data. This structure is variable in size and the user must ensure enough memory has been allocated for it.

---

## mmioCFChangeEntry Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input  
Specifies options for the operation. Contains none or the following flag:

**MMIO\_CHANGEDELETED**  
Allows a deleted CTOC entry to be changed.

---

## mmioCFChangeEntry Return Value - rc

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure:

**MMIO\_CF\_SUCCESS**  
If the function succeeds, 0 is returned.

**MMIOERR\_INVALID\_HANDLE**  
The handle passed was not valid.

**MMIOERR\_INVALID\_PARAMETER**  
The parameter passed was not correct. For this function, a *pmmctocentry* NULL is invalid.

**MMIOERR\_READ\_ONLY\_FILE**  
The RIFF compound-file is opened as read-only.

**MMIO\_CF\_FAILURE**  
The function failed. The *ulErrorRet* field of the **MMIOINFO** structure contains additional information as follows:

**MMIOERR\_CF\_ENTRY\_NOT\_FOUND**  
The CTOC entry corresponding to the element was not found.

**MMIOERR\_INTERNAL\_SYSTEM**  
The operation failed due to an internal system error.

-----

## mmioCFChangeEntry - Parameters

**hmmcf** (**HMMCF**) - input  
A RIFF compound-file handle returned by **mmioCFOpen**.

**pmmctocentry** (**PMMCTOCENTRY**) - input  
A pointer to the **MMCTOCENTRY** structure containing the modified CTOC data. This structure is variable in size and the user must ensure enough memory has been allocated for it.

**ulFlags** (**ULONG**) - input  
Specifies options for the operation. Contains none or the following flag:

**MMIO\_CHANGEDELETED**  
Allows a deleted CTOC entry to be changed.

**rc** (**ULONG**) - returns  
Return codes indicating success or type of failure:

**MMIO\_CF\_SUCCESS**  
If the function succeeds, 0 is returned.

**MMIOERR\_INVALID\_HANDLE**  
The handle passed was not valid.

**MMIOERR\_INVALID\_PARAMETER**  
The parameter passed was not correct. For this function, a *pmmctocentry* NULL is invalid.

**MMIOERR\_READ\_ONLY\_FILE**  
The RIFF compound-file is opened as read-only.

**MMIO\_CF\_FAILURE**  
The function failed. The *ulErrorRet* field of the **MMIOINFO** structure contains additional information as follows:

**MMIOERR\_CF\_ENTRY\_NOT\_FOUND**  
The CTOC entry corresponding to the element was not found.

**MMIOERR\_INTERNAL\_SYSTEM**  
The operation failed due to an internal system error.

-----

## mmioCFChangeEntry - Remarks

The identifier for the entry is the element name, which is appended to the end of the **MMCTOCENTRY** structure passed in the

*pmmtocentry* parameter. (The element name itself can not be modified.) `mmioCFChangeEntry` updates the compound-file CTOC entry with the information contained `MMCTOCENTRY` structure. If the compression technique is changed, the *ulUncompressBytes* field of the `MMCTOCENTRY` structure must also be changed. When the compression technique is NULL, the *ulUncompressBytes* field must be the size in bytes of the element when it is uncompressed. Consider calling `mmioCFFindEntry` to fill in the `MMCTOCENTRY` structure prior to calling this function.

---

## mmioCFChangeEntry - Related Functions

- [mmioCFAddEntry](#)
  - [mmioCFFindEntry](#)
  - [mmioCFDeleteEntry](#)
- 

## mmioCFChangeEntry - Example Code

The following code illustrates how to modify an entry.

```
HMMCF hmmcfl;
MMCTOCENTRY mmctocentry;
ULONG ulFlags;
ULONG rc;
...

memset( &mmctocentry, '\\0', sizeof(MMCTOCENTRY));
rc = mmioCFChangeEntry( hmmcfl, &mmctocentry, ulFlags);
if (rc)
    /* error */
else
    ...
```

---

## mmioCFChangeEntry - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioCFClose

---

## mmioCFClose - Syntax

This function closes a compound file that was opened by [mmioCFOpen](#).

```
#define INCL_MMIOOS2
#include <os2.h>

HMMCF    hmmcf;    /* Compound-file handle. */
ULONG    ulFlags;   /* Reserved. */
ULONG    rc;        /* Return codes. */

rc = mmioCFClose(hmmcf, ulFlags);
```

---

## mmioCFClose Parameter - hmmcf

**hmmcf** ([HMMCF](#)) - input  
A RIFF compound-file handle returned by [mmioCFOpen](#).

---

## mmioCFClose Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input  
Reserved for future use and must be set to zero.

---

## mmioCFClose Return Value - rc

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure:

- MMIO\_CF\_SUCCESS  
If the function succeeds, 0 is returned.
- MMIOERR\_INVALID\_HANDLE  
The handle passed was not valid.
- MMIO\_CF\_FAILURE  
The function failed. A call to [mmioGetLastError](#) might return one of the following errors:
  - MMIOERR\_CF\_NON\_BND\_FILE  
Tried to close a non RIFF compound file.
  - MMIOERR\_CF\_ELEMENTS\_OPEN  
Compound-file elements are open.
  - MMIOERR\_INTERNAL\_SYSTEM  
The operation failed due to an internal system error.

---

## mmioCFClose - Parameters



**hmmcf** ([HMMCF](#)) - input

A RIFF compound-file handle returned by [mmioCFOpen](#).

**ulFlags** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_CF\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIO\_CF\_FAILURE

The function failed. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_CF\_NON\_BND\_FILE

Tried to close a non RIFF compound file.

MMIOERR\_CF\_ELEMENTS\_OPEN

Compound-file elements are open.

MMIOERR\_INTERNAL\_SYSTEM

The operation failed due to an internal system error.

-----

## mmioCFClose - Remarks

This function decrements the usage count of the compound-file table of contents (CTOC) maintained in memory for this RIFF compound file. If the usage count drops to 0, the CTOC is written to disk, and the RIFF compound-file handle is closed. An `mmioCFClose` operation fails if there are any open elements for this RIFF compound file. The user will get an error if [mmioClose](#) is used instead of `mmioCFClose` when trying to close a RIFF compound file.

If this is an *ExitList* close, all open elements are closed, and the `mmioCFClose` operation is allowed. If the compound file was opened as read-only, the CTOC will not be rewritten.

If the `mmioCFClose` function fails and the user had modified compound-file-resource-group (CGRP) elements, the data stored on the file is inconsistent. To correct the problem, the user must create free file space and attempt to close again.

-----

## mmioCFClose - Related Functions

- [mmioCFOpen](#)

-----

## mmioCFClose - Example Code

The following code illustrates how to close a file.

```
HMMCF hmmcf1;  
ULONG ulFlags;  
ULONG rc;
```

```
...

rc = mmioCFClose( hmmcfl, ulFlags);
if (rc)
    /* error */
else
    ...
```

---

## mmioCFClose - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

---

## mmioCFCompact

---

## mmioCFCompact - Syntax

This function compacts the elements of a RIFF compound file.

```
#define INCL_MMIOOS2
#include <os2.h>

PSZ      pszFileName; /* RIFF file name. */
ULONG    ulFlags;      /* Reserved. */
ULONG    rc;           /* Return codes. */

rc = mmioCFCompact(pszFileName, ulFlags);
```

---

## mmioCFCompact Parameter - pszFileName

**pszFileName** ([PSZ](#)) - input

The name of the RIFF compound file to compact.

---

## mmioCFCompact Parameter - ulFlags

**ulFlags** (**ULONG**) - input  
Reserved for future use and must be set to zero.

---

## mmioCFCompact Return Value - rc

**rc** (**ULONG**) - returns  
Return codes indicating success or type of failure:

- MMIO\_SUCCESS**  
If the function succeeds, 0 is returned.
  - MMIO\_ERROR**  
The open failed. Possible reasons for failure include a nonexistent file name or an already open file.
  - MMIOERR\_INVALID\_FILENAME**  
No file name was specified.
  - MMIOERR\_INTERNAL\_SYSTEM**  
An internal system error has occurred.
  - MMIOERR\_NO\_CORE**  
The buffer cannot be allocated.
- 

## mmioCFCompact - Parameters

**pszFileName** (**PSZ**) - input  
The name of the RIFF compound file to compact.

**ulFlags** (**ULONG**) - input  
Reserved for future use and must be set to zero.

**rc** (**ULONG**) - returns  
Return codes indicating success or type of failure:

- MMIO\_SUCCESS**  
If the function succeeds, 0 is returned.
  - MMIO\_ERROR**  
The open failed. Possible reasons for failure include a nonexistent file name or an already open file.
  - MMIOERR\_INVALID\_FILENAME**  
No file name was specified.
  - MMIOERR\_INTERNAL\_SYSTEM**  
An internal system error has occurred.
  - MMIOERR\_NO\_CORE**  
The buffer cannot be allocated.
- 

## mmioCFCompact - Remarks

This function is useful for writers of audio-enabling macros who use compound files in their implementation.

The mmioCFCompact function does not use a compression algorithm to compact the compound file. It merely deletes elements in the CGRP whose *ulMedType* field of [MMCTOCENTRY](#) are marked as FOURCC\_DEL and updates the *ulMedType* field to FOURCC\_FREE. At the completion of the function, CTOC entries are sorted according to offset. Entries might not remain sorted if subsequent appends are made.

The mmioCFCompact function compacts the file in place; that is, the function rewrites the file within the same buffer as the source to save memory. No original can be salvaged if an error occurs during compaction. If this behavior is unacceptable use [mmioCFCopy](#).

The [mmioCFCopy](#) function also compacts a file, but it rewrites the file to a specified target name. The target name cannot be the same as the source file name. Therefore, with this function you must delete the source file.

---

## mmioCFCompact - Related Functions

- [mmioCFCopy](#)

---

## mmioCFCompact - Example Code

The following code illustrates compacting a file with mmioCFCompact.

```
PSZ pszFileName;  
ULONG ulFlags ;  
ULONG ulReturnCode;  
  
pszFileName = "SOUNDS.BND";  
ulFlags = 0L;  
  
ulReturnCode = mmioCFCompact( pszFileName, ulFlags );  
  
if (ulReturnCode)  
    /* error */  
else  
    /* success */
```

---

## mmioCFCompact - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioCFCopy

---

## mmioCFCopy - Syntax

This function copies the compound-file table of contents (CTOC) and the compound-file resource group (CGRP) chunks from an open RIFF compound file to another RIFF compound file. The newly written compound-file resource group (CGRP) chunk will be compacted; that is, it will have no deleted elements.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMCF    hmmcfSource;    /* RIFF file handle. */
PSZ      pszDestFileName; /* Pointer to destination file name. */
ULONG    ulFlags;        /* Reserved. */
ULONG    rc;             /* Return codes. */

rc = mmioCFCopy(hmmcfSource, pszDestFileName,
                ulFlags);
```

---

## mmioCFCopy Parameter - hmmcfSource

**hmmcfSource** ([HMMCF](#)) - input  
A RIFF compound-file handle returned by [mmioCFOpen](#). This is the source file to be copied.

---

## mmioCFCopy Parameter - pszDestFileName

**pszDestFileName** ([PSZ](#)) - input  
The pointer to the name of the destination file. The RIFF compound file is copied to the destination file.

---

## mmioCFCopy Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input  
Reserved for future use and must be set to zero.

---

## mmioCFCopy Return Value - rc

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_CF\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

The file name was null or tried to copy the file to itself.

MMIOERR\_READ\_ONLY\_FILE

The RIFF compound file is opened as read-only.

MMIO\_CF\_FAILURE

The operation failed due to an internal system error. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_CF\_ELEMENTS\_OPEN

Compound-file elements are open.

MMIOERR\_NO\_CORE

Not enough memory is available.

MMIOERR\_INTERNAL\_SYSTEM

The operation failed due to an internal system error.

-----

## mmioCFCopy - Parameters

**hmmcfSource** ([HMMCF](#)) - input

A RIFF compound-file handle returned by [mmioCFOpen](#). This is the source file to be copied.

**pszDestFileName** ([PSZ](#)) - input

The pointer to the name of the destination file. The RIFF compound file is copied to the destination file.

**ulFlags** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_CF\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

The file name was null or tried to copy the file to itself.

MMIOERR\_READ\_ONLY\_FILE

The RIFF compound file is opened as read-only.

MMIO\_CF\_FAILURE

The operation failed due to an internal system error. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_CF\_ELEMENTS\_OPEN

Compound-file elements are open.

MMIOERR\_NO\_CORE

Not enough memory is available.

MMIOERR\_INTERNAL\_SYSTEM

The operation failed due to an internal system error.

---

## mmioCFCopy - Remarks

mmioCFCopy copies the CTOC and CGRP chunks of an open source RIFF compound file to the target file. Deleted elements are not copied to the new file. mmioCFCopy opens the target file (using [mmioOpen](#) with the MMIO\_CREATE flag) and builds a RIFF BND header at the beginning of the file. The CTOC and CGRP chunks then are copied. Notice that it is invalid to copy the RIFF compound file to itself. Upon completion of the copy operation, the target file is closed and resides on the file system, while the source file is unaltered. The target file must not be opened before mmioCFCopy is called.

As a note for performance considerations, the function either copies the entire CTOC and CGRP chunks in one single system buffer, or does a fixed page-size copy. The method used depends on the size of the RIFF compound file and is determined by the system. If the copy operation fails, the target file is deleted.

If the target already exists, it is overwritten by the copy operation.

---

## mmioCFCopy - Related Functions

- [mmioCFAddElement](#)
- 

## mmioCFCopy - Example Code

The following code illustrates how to copy from a file.

```
HMMCF hmmcfSource;
PSZ pszDestFileName;
ULONG ulFlags;
ULONG rc;
...

rc = mmioCFCopy( hmmcfSource, pszDestFileName, ulFlags);
if (rc)
    /* error */
else
    ...
```

---

## mmioCFCopy - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

# mmioCFDeleteEntry

---

## mmioCFDeleteEntry - Syntax

This function deletes a compound-file table of contents (CTOC) entry from an open RIFF compound file.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMCF      hmmcf;          /* RIFF file handle. */
PMMCTOCENTRY pmmctocentry; /* Pointer to MMCTOCENTRY data structure. */
ULONG      ulFlags;        /* Reserved. */
ULONG      rc;             /* Return codes. */

rc = mmioCFDeleteEntry(hmmcf, pmmctocentry,
    ulFlags);
```

---

## mmioCFDeleteEntry Parameter - hmmcf

**hmmcf** ([HMMCF](#)) - input  
A RIFF compound-file handle returned by [mmioCFOpen](#).

---

## mmioCFDeleteEntry Parameter - pmmctocentry

**pmmctocentry** ([PMMCTOCENTRY](#)) - input  
A pointer to the [MMCTOCENTRY](#) data structure containing the RIFF compound-file element name. This structure is variable in size, and the user must ensure enough memory has been allocated for it.

---

## mmioCFDeleteEntry Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input  
Reserved for future use and must be set to zero.

---

## mmioCFDeleteEntry Return Value - rc



**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_CF\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

For this function, a *pszElementName*, *pchBuffer*, or *cchBytes* NULL is invalid.

MMIOERR\_READ\_ONLY\_FILE

The RIFF compound file is opened as read-only.

MMIO\_CF\_FAILURE

The function failed. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_CF\_ENTRY\_NOT\_FOUND

The CTOC entry corresponding to the element was not found.

MMIOERR\_INTERNAL\_SYSTEM

The operation failed due to an internal system error.

-----

## mmioCFDeleteEntry - Parameters

**hmmcf** ([HMMCF](#)) - input

A RIFF compound-file handle returned by [mmioCFOpen](#).

**pmmctocentry** ([PMMCTOCENTRY](#)) - input

A pointer to the [MMCTOCENTRY](#) data structure containing the RIFF compound-file element name. This structure is variable in size, and the user must ensure enough memory has been allocated for it.

**ulFlags** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_CF\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

For this function, a *pszElementName*, *pchBuffer*, or *cchBytes* NULL is invalid.

MMIOERR\_READ\_ONLY\_FILE

The RIFF compound file is opened as read-only.

MMIO\_CF\_FAILURE

The function failed. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_CF\_ENTRY\_NOT\_FOUND

The CTOC entry corresponding to the element was not found.

MMIOERR\_INTERNAL\_SYSTEM

The operation failed due to an internal system error.

-----

## mmioCFDeleteEntry - Remarks

The identifier for the entry is the element name, which is appended to the end of the [MMCTOCENTRY](#) structure passed in the *pmmctocentry* parameter. The CTOC entry is marked as deleted by setting the *ulMedType* field to FOURCC\_DEL. The actual element data remains in place. To remove data that was previously marked for deletion, use [mmioCFCopy](#). The result will be a compressed file with all those elements marked for deletion physically removed.

---

## mmioCFDeleteEntry - Related Functions

- [mmioCFAddEntry](#)
  - [mmioCFChangeEntry](#)
  - [mmioCFFindEntry](#)
- 

## mmioCFDeleteEntry - Example Code

The following code illustrates how to delete an entry.

```
HMMCF hmmcfl;
MMCTOCENTRY &mmctocentry;
ULONG ulFlags;
ULONG rc;
...

memset( &mmctocentry, '\\0', sizeof(MMCTOCENTRY));
rc = mmioCFDeleteEntry( hmmcfl, &mmctocentry, ulFlags);
if (rc)
    /* error */
else
    ...
```

---

## mmioCFDeleteEntry - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioCFFindEntry

---

## mmioCFFindEntry - Syntax

This function finds a particular CTOC entry in an open RIFF compound file.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMCF      hmmcf;          /* RIFF file handle. */
PMMCTOCENTRY pmmctocentry; /* Pointer to MMCTOCENTRY. */
ULONG      ulFlags;        /* Flags. */
ULONG      rc;             /* Return codes. */

rc = mmioCFFindEntry(hmmcf, pmmctocentry,
                    ulFlags);
```

---

## mmioCFFindEntry Parameter - hmmcf

**hmmcf** ([HMMCF](#)) - input

A RIFF compound-file handle returned by [mmioCFOpen](#).

---

## mmioCFFindEntry Parameter - pmmctocentry

**pmmctocentry** ([PMMCTOCENTRY](#)) - in/out

A pointer to the [MMCTOCENTRY](#) structure containing the name of the RIFF compound-file element to search for. This structure is variable in size and the user must ensure enough memory has been allocated for it. Flags in *ulFlags* can be set to specify that an element is to be searched for by some attribute other than name.

---

## mmioCFFindEntry Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input

This parameter can be used to specify that an element is to be searched for by some attribute other than name. The MMIO\_FINDFIRST and MMIO\_FINDNEXT flags are mutually exclusive. An MMIOERR\_CF\_ENTRY\_NOT\_USED error is returned if a matching entry is not found or if MMIO\_FINDNEXT was specified and no more entries match the search CTOC entry.

The following flags are supported:

MMIO\_FINDFIRST

Find the first entry in the CTOC table.

MMIO\_FINDNEXT

Find the next entry in the CTOC table after the entry previously found and returned in the *pmmctocentry* parameter. The *pmmctocentry* parameter must contain the previous CTOC entry. Returns NULL if *pmmctocentry* refers to the last entry.

MMIO\_FINDDELETED

Find an entry in the CTOC table that has been marked as deleted

MMIO\_FINDUNUSED

Find an entry in the CTOC table that has been marked as unused. A default compound file contains 16 unused CTOC entries in the CTOC table. As each CTOC entry and element is added, one of these unused entries is used.

-----

## mmioCFFindEntry Return Value - rc

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_CF\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

For this function a *pszElementName*, *pchBuffer*, or *cchBytes* NULL is invalid.

MMIOERR\_READ\_ONLY\_FILE

The RIFF compound-file is opened as read-only.

MMIOERR\_CF\_ENTRY\_NOT\_FOUND

System failed to find CTOC entry.

MMIO\_CF\_FAILURE

The function failed. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_WRITE\_ONLY\_FILE

File not opened in read mode.

MMIOERR\_INTERNAL\_SYSTEM

The operation failed due to an internal system error.

-----

## mmioCFFindEntry - Parameters

**hmmcf** ([HMMCF](#)) - input

A RIFF compound-file handle returned by [mmioCFOpen](#).

**pmmctocentry** ([PMMCTOCENTRY](#)) - in/out

A pointer to the [MMCTOCENTRY](#) structure containing the name of the RIFF compound-file element to search for. This structure is variable in size and the user must ensure enough memory has been allocated for it. Flags in *ulFlags* can be set to specify that an element is to be searched for by some attribute other than name.

**ulFlags** ([ULONG](#)) - input

This parameter can be used to specify that an element is to be searched for by some attribute other than name. The MMIO\_FINDFIRST and MMIO\_FINDNEXT flags are mutually exclusive. An MMIOERR\_CF\_ENTRY\_NOT\_USED error is returned if a matching entry is not found or if MMIO\_FINDNEXT was specified and no more entries match the search CTOC entry.

The following flags are supported:

MMIO\_FINDFIRST

Find the first entry in the CTOC table.

MMIO\_FINDNEXT

Find the next entry in the CTOC table after the entry previously found and returned in the *pmmctocentry*

parameter. The *pmmctocentry* parameter must contain the previous CTOC entry. Returns NULL if *pmmctocentry* refers to the last entry.

#### MMIO\_FINDDELETED

Find an entry in the CTOC table that has been marked as deleted

#### MMIO\_FINDUNUSED

Find an entry in the CTOC table that has been marked as unused. A default compound file contains 16 unused CTOC entries in the CTOC table. As each CTOC entry and element is added, one of these unused entries is used.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

#### MMIO\_CF\_SUCCESS

If the function succeeds, 0 is returned.

#### MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

#### MMIOERR\_INVALID\_PARAMETER

For this function a *pszElementName*, *pchBuffer*, or *cchBytes* NULL is invalid.

#### MMIOERR\_READ\_ONLY\_FILE

The RIFF compound-file is opened as read-only.

#### MMIOERR\_CF\_ENTRY\_NOT\_FOUND

System failed to find CTOC entry.

#### MMIO\_CF\_FAILURE

The function failed. A call to [mmioGetLastError](#) might return one of the following errors:

#### MMIOERR\_WRITE\_ONLY\_FILE

File not opened in read mode.

#### MMIOERR\_INTERNAL\_SYSTEM

The operation failed due to an internal system error.

-----

## mmioCFFindEntry - Remarks

The search is not case-sensitive; the flags operate as follows:

- MMIO\_FINDFIRST and MMIO\_FINDNEXT cannot be specified together.

If an element is specified, it is ignored; if no element is specified, the flags operate as follows:

- If MMIO\_FINDFIRST is specified, the first non-deleted entry is returned.
- If MMIO\_FINDFIRST and MMIO\_FINDDELETED are specified, the first deleted element is returned.

All other cases use the element name in the search, and the flags operate as follows:

- If no flags are specified, the first non-deleted entry that matches the element name passed in is returned.
- If MMIO\_FINDNEXT is specified, the entry that matches the element name passed in is found. The first non-deleted entry following this entry is returned.
- If MMIO\_FINDDELETED is specified, the entry that matches the element name passed in is found. If the entry is marked deleted, it is returned.
- If MMIO\_FINDNEXT and MMIO\_FINDDELETED are specified, the entry that matches the element name passed in is found. The element name passed in may refer to an existing or deleted entry. At this point, the next entry that is deleted is returned.

If the function succeeds, the [MMCTOCENTRY](#) structure passed in the *pmmctocentry* parameter is filled in with information about the CTOC entry. The user can proceed through the CTOC entry list by using MMIO\_FINDFIRST followed by a series of MMIO\_FINDNEXT actions, using the information from the previous call.

-----

## mmioCFFindEntry - Related Functions

- [mmioCFAddEntry](#)
- [mmioCFChangeEntry](#)
- [mmioCFDeleteEntry](#)

---

## mmioCFFindEntry - Example Code

The following code illustrates how to find an entry.

```
HMMCF hmmcfl;
MMCTOCENTRY mmctocentry;
ULONG ulFlags;
ULONG rc;
...

memset( &mmctocentry, '\0', sizeof(MMCTOCENTRY));
rc = mmioCFFindEntry( hmmcfl, &mmctocentry, ulFlags);
if (rc)
    /* error */
else
    ...
```

---

## mmioCFFindEntry - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioCFGetInfo

---

## mmioCFGetInfo - Syntax

This function retrieves the compound-file table of contents (CTOC) header of an open RIFF compound file.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMCF hmmcfl; /* RIFF file handle. */
```

```
PMMCFINFO    pmmcfinfo; /* Pointer to MMCFINFO. */
ULONG        cBytes;    /* Buffer size. */
ULONG        rc;        /* Return codes. */

rc = mmioCFGetInfo(hmmcf, pmmcfinfo, cBytes);
```

---

## mmioCFGetInfo Parameter - hmmcf

**hmmcf** ([HMMCF](#)) - input  
A RIFF compound-file handle returned by [mmioCFOpen](#).

---

## mmioCFGetInfo Parameter - pmmcfinfo

**pmmcfinfo** ([PMMCFINFO](#)) - in/out  
A pointer to the [MMCFINFO](#) data structure, which is filled with the CTOC header. This structure is variable in size, and the user must ensure enough memory has been allocated for it.

---

## mmioCFGetInfo Parameter - cBytes

**cBytes** ([ULONG](#)) - input  
Size of the buffer that the *pmmcfinfo* parameter points to. This is the maximum number of bytes that will be copied.

---

## mmioCFGetInfo Return Value - rc

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure. If the function succeeds, the number of bytes copied is returned. A NULL is returned for a failure. Additional error information is stored in the *ulErrorRet* field of [MMIOINFO](#) as follows:

- MMIOERR\_INVALID\_PARAMETER  
The parameter passed was invalid. For this function, a NULL *pmmcfinfo* is invalid. *cBytes* must be greater than 0.
- MMIOERR\_WRITE\_ONLY\_FILE  
File not opened in read mode.
- MMIOERR\_INTERNAL\_SYSTEM  
The operation failed due to an internal system error.

---

## mmioCFGetInfo - Parameters

**hmmcf** (**HMMCF**) - input

A RIFF compound-file handle returned by [mmioCFOpen](#).

**pmmcfinfo** (**PMMCFINFO**) - in/out

A pointer to the [MMCFINFO](#) data structure, which is filled with the CTOC header. This structure is variable in size, and the user must ensure enough memory has been allocated for it.

**cBytes** (**ULONG**) - input

Size of the buffer that the *pmmcfinfo* parameter points to. This is the maximum number of bytes that will be copied.

**rc** (**ULONG**) - returns

Return codes indicating success or type of failure. If the function succeeds, the number of bytes copied is returned. A NULL is returned for a failure. Additional error information is stored in the *ulErrorRet* field of [MMIOINFO](#) as follows:

MMIOERR\_INVALID\_PARAMETER

The parameter passed was invalid. For this function, a NULL *pmmcfinfo* is invalid. *cBytes* must be greater than 0.

MMIOERR\_WRITE\_ONLY\_FILE

File not opened in read mode.

MMIOERR\_INTERNAL\_SYSTEM

The operation failed due to an internal system error.

---

## mmioCFGetInfo - Remarks

The information copied to the *pmmcfinfo* parameter consists of an [MMCFINFO](#) structure followed by the variable-length arrays: *aulExHdrFldUsage*, *aulExEntFldUsage*, and *aulExHdrField*.

To find out how large a buffer the user needs to allocate, call [mmioCFGetInfo](#) with the *cBytes* equal to the size of a ULONG. This will return the first field of the CTOC header, which happens to be the size of the header. This size can be used as *cBytes* on the subsequent call.

---

## mmioCFGetInfo - Related Functions

- [mmioCFSetInfo](#)

---

## mmioCFGetInfo - Example Code

The following code illustrates how to retrieve information from a file.

```
HMMCF hmmcf1;
MMCFINFO &mmmcfinfo;
ULONG cBytes;
ULONG rc;
...

memset(&mmmcfinfo, '\0', sizeof(MMCFINFO));
rc = mmioCFGetInfo( hmmcf1, &mmmcfinfo, (ULONG)sizeof(ULONG));
if (rc != (ULONG)sizeof(ULONG))
    break;
else
    cBytes = pmmcfinfo->ulHeaderSize;
rc = mmioCFGetInfo( hmmcf1, mmmcfinfo, cBytes);
```



```
if (rc)
    /* error */
else
    ...
```

---

## mmioCFGetInfo - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

---

## mmioCFOpen

---

## mmioCFOpen - Syntax

This function opens a RIFF compound file by name.

```
#define INCL_MMIOOS2
#include <os2.h>

PSZ      pszFileName; /* RIFF file name. */
PMMCFINFO pmmcfinfo; /* Pointer to MMCFINFO. */
PMMIOINFO pmmioinfo; /* Pointer to MMIOINFO. */
ULONG     ulFlags;    /* Flags. */
HMMCF     hmmcf;      /* RIFF compound-file handle. */

hmmcf = mmioCFOpen(pszFileName, pmmcfinfo,
                  pmmioinfo, ulFlags);
```

---

## mmioCFOpen Parameter - pszFileName

**pszFileName** ([PSZ](#)) - input

The name of the RIFF compound file to open.

---

## mmioCFOpen Parameter - pmmcfinfo

**pmmcfinfo** ([PMMCFINFO](#)) - input

A pointer to the [MMCFINFO](#) data structure containing optional header information. It can be NULL.

---

## mmioCFOpen Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#)) - input

A pointer to the [MMIOINFO](#) information structure containing optional open information that is passed to [mmioOpen](#). It can be NULL.

---

## mmioCFOpen Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input

Contains none or some of the following flags. The MMIO\_READ, MMIO\_WRITE, and MMIO\_READWRITE flags are mutually exclusive.

MMIO\_READ

Open the file for reading only. This is the default if MMIO\_WRITE and MMIO\_READWRITE are not specified.

MMIO\_WRITE

Open the file for writing. A file cannot be read from if the file is opened in this mode.

MMIO\_READWRITE

Open the file for both reading and writing.

MMIO\_EXCLUSIVE

Open the file with exclusive mode, denying other processes both read and write access to the file. mmioCFOpen fails if the file has been opened in any other mode for read or write access, even by the current process.

MMIO\_DENYWRITE

Open the file and deny other processes write access to the file. mmioCFOpen fails if the file has been opened by a compatible process or for write access by any other process.

MMIO\_DENYREAD

Open the file and deny other processes read access to the file. mmioCFOpen fails if the file has been opened by a compatible process or for read access by any other process.

MMIO\_DENYNONE

Open the file and deny other processes read access to the file. mmioCFOpen fails if the file has been opened by a compatible process or for read access by any other process.

MMIO\_CREATE

Directs mmioCFOpen to create a new file. If the file already exists, it is truncated to 0 length, unless it is already opened. In that case, a handle ([HMMCF](#)) to the RIFF compound file is returned.

---

## mmioCFOpen Return Value - hmmcf

**hmmcf** ([HMMCF](#)) - returns

If the function succeeds, the handle to the RIFF compound file is returned. A NULL is returned if it fails.

---

## mmioCFOpen - Parameters

**pszFileName** ([PSZ](#)) - input

The name of the RIFF compound file to open.

**pmmcfinfo** ([PMMCFINFO](#)) - input

A pointer to the [MMCFINFO](#) data structure containing optional header information. It can be NULL.

**pmmioinfo** ([PMMIOINFO](#)) - input

A pointer to the [MMIOINFO](#) information structure containing optional open information that is passed to [mmioOpen](#). It can be NULL.

**ulFlags** ([ULONG](#)) - input

Contains none or some of the following flags. The MMIO\_READ, MMIO\_WRITE, and MMIO\_READWRITE flags are mutually exclusive.

MMIO\_READ

Open the file for reading only. This is the default if MMIO\_WRITE and MMIO\_READWRITE are not specified.

MMIO\_WRITE

Open the file for writing. A file cannot be read from if the file is opened in this mode.

MMIO\_READWRITE

Open the file for both reading and writing.

MMIO\_EXCLUSIVE

Open the file with exclusive mode, denying other processes both read and write access to the file. mmioCFOpen fails if the file has been opened in any other mode for read or write access, even by the current process.

MMIO\_DENYWRITE

Open the file and deny other processes write access to the file. mmioCFOpen fails if the file has been opened by a compatible process or for write access by any other process.

MMIO\_DENYREAD

Open the file and deny other processes read access to the file. mmioCFOpen fails if the file has been opened by a compatible process or for read access by any other process.

MMIO\_DENYNONE

Open the file and deny other processes read access to the file. mmioCFOpen fails if the file has been opened by a compatible process or for read access by any other process.

MMIO\_CREATE

Directs mmioCFOpen to create a new file. If the file already exists, it is truncated to 0 length, unless it is already opened. In that case, a handle ([HMMCF](#)) to the RIFF compound file is returned.

**hmmcf** ([HMMCF](#)) - returns

If the function succeeds, the handle to the RIFF compound file is returned. A NULL is returned if it fails.

---

## mmioCFOpen - Remarks

This function will either construct a compound-file table of contents (CTOC) in memory for this compound file or give the caller access to a CTOC that already exists in memory for this compound file. Only one CTOC for a particular compound file is maintained in memory at any given time. This CTOC is shared by any process that needs access to the file.

This function will determine if the CTOC for this compound file is being maintained in memory. If so, the access and sharing modes are checked to ensure that an open operation is allowed. The existing handle ([HMMCF](#)) for the CTOC is returned to the caller. If the file had not been previously opened, this function will construct a CTOC in memory for this file. If the name is not pointing to a valid BND file, an error is returned.

The RIFF compound-file name cannot contain a + or | because they are used to express elements of compound files.

The access and sharing flags are maintained only within the set of compound-file functions. If the RIFF compound file or elements are accessed without using the compound-file calls, the access and sharing modes are unpredictable. An [mmioOpen](#) operation with a fully qualified element name is considered a compound-file call, because it internally calls mmioCFOpen; thus the flags are predictable in that case.

Access and sharing modes are supported for compound files. However, these modes are enforced only within the MMIO services compound-file functions and the [mmioOpen](#) of compound-file elements. The sharing modes work as in the base operating system except that elements ignore the sharing mode of the RIFF compound file. Elements do obey the access modes of the RIFF compound file.

Elements can be opened and used from the compound file by sending the element name that is stored in the CTOC to the [mmioOpen](#) function.

The FOURCC of FOURCC\_BND should be used only for the element and not the compound file itself. That is, do *not* specify FOURCC\_BND when using mmioCFOpen.

---

## mmioCFOpen - Related Functions

- [mmioCFClose](#)

---

## mmioCFOpen - Example Code

The following code illustrates how to open a file.

```
HMMCF hmmcfl;
MMCFINFO mmcfinfo;
MMIOINFO mmioinfo;
ULONG ulFlags;
...

memset( &mmcfinfo, '\0', sizeof(MMCFINFO));
memset( &mmioinfo, '\0', sizeof(MMIOINFO));
strcpy( pFileName, "myfile.bnd" );
ulFlags = 0L;

hmmcfl = mmioCFOpen( pFileName, &mmcfinfo, &mmioinfo, ulFlags)
if (!hmmcfl)
    /* error */
else
    ...
```

---

## mmioCFOpen - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

# mmioCFSetInfo

---

## mmioCFSetInfo - Syntax

This function modifies information that is stored in the compound-file table of contents (CTOC) header of an open RIFF compound file.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMCF      hmmcf;      /* RIFF file handle. */
PMMCFINFO  pmmcfinfo;  /* Pointer to MMCFINFO. */
ULONG      cBytes;      /* Max # of bytes. */
ULONG      rc;          /* Return codes. */

rc = mmioCFSetInfo(hmmcf, pmmcfinfo, cBytes);
```

---

## mmioCFSetInfo Parameter - hmmcf

**hmmcf** ([HMMCF](#)) - input

A pointer to a user-supplied buffer that contains the modified CTOC header. A RIFF compound-file handle is returned by [mmioCFOpen](#).

---

## mmioCFSetInfo Parameter - pmmcfinfo

**pmmcfinfo** ([PMMCFINFO](#)) - input

A pointer to the [MMCFINFO](#) data structure that contains the modified CTOC header. This buffer was filled in by [mmioCFGetInfo](#) and then modified by the user.

---

## mmioCFSetInfo Parameter - cBytes

**cBytes** ([ULONG](#)) - input

Size of the buffer that the *pmmcfinfo* parameter points to. This is the maximum number of bytes that will be copied.

---

## mmioCFSetInfo Return Value - rc

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure. If the function succeeds, the number of bytes copied is returned. A NULL is returned for a failure. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_INVALID\_PARAMETER

For this function, a NULL *pmmcfinfo* is invalid. *cBytes* must be greater than 0.

MMIOERR\_READ\_ONLY\_FILE

File not opened in write mode.

MMIOERR\_INTERNAL\_SYSTEM

The operation failed due to an internal system error.

-----

## mmioCFSetInfo - Parameters

**hmmcf** ([HMMCF](#)) - input

A pointer to a user-supplied buffer that contains the modified CTOC header. A RIFF compound-file handle is returned by [mmioCFOpen](#).

**pmmcfinfo** ([PMMCFINFO](#)) - input

A pointer to the [MMCFINFO](#) data structure that contains the modified CTOC header. This buffer was filled in by [mmioCFGetInfo](#) and then modified by the user.

**cBytes** ([ULONG](#)) - input

Size of the buffer that the *pmmcfinfo* parameter points to. This is the maximum number of bytes that will be copied.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure. If the function succeeds, the number of bytes copied is returned. A NULL is returned for a failure. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_INVALID\_PARAMETER

For this function, a NULL *pmmcfinfo* is invalid. *cBytes* must be greater than 0.

MMIOERR\_READ\_ONLY\_FILE

File not opened in write mode.

MMIOERR\_INTERNAL\_SYSTEM

The operation failed due to an internal system error.

-----

## mmioCFSetInfo - Remarks

The only data that should be modified by the user is *aulExHdrFldUsage* and *aulExHdrField* fields appended to the end of the [MMIOINFO](#) structure passed in the *pmmcfinfo* parameter.

-----

## mmioCFSetInfo - Related Functions

- [mmioCFGetInfo](#)

-----

# mmioCFSetInfo - Example Code

The following code illustrates how to modify information in a file.

```
HMMCF hmmcfl;
MMCFINFO &mmcfinfo;
ULONG cBytes;
ULONG rc;
...

memset(&mmcfinfo, '\0', sizeof(MMCFINFO));
rc = mmioCFGetInfo( hmmcfl, &mmcfinfo, (ULONG)sizeof(ULONG));
if (rc != (ULONG)sizeof(ULONG))
    break;
else
    cBytes = pmmcfinfo->ulHeaderSize;
...

rc = mmioCFSetInfo( hmmcfl, &mmcfinfo, cBytes);
if (rc)
    /* error */
else
    ...
```

---

## mmioCFSetInfo - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioClose

---

## mmioClose - Syntax

This function closes a file that was opened by [mmioOpen](#).

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO    hmmio;    /* Open file handle. */
USHORT    usFlags; /* Flags. */
USHORT    rc;      /* Return codes. */

rc = mmioClose(hmmio, usFlags);
```

---

## mmioClose Parameter - hmmio

**hmmio** ([HMMIO](#)) - input  
The open file handle returned by [mmioOpen](#).

---

## mmioClose Parameter - usFlags

**usFlags** ([USHORT](#)) - input  
Contains nothing or the following flag:

**MMIO\_FHOPEN**  
This flag is used to tell the I/O to not close the file or files of type FOURCC\_DOS. This allows an [HMMIO](#) instance to be closed in cases where a DOS file handle was provided to the I/O procedure on an [mmioOpen](#) call.

---

## mmioClose Return Value - rc

**rc** ([USHORT](#)) - returns  
Return codes indicating success or type of failure:

**MMIO\_SUCCESS**  
If the function succeeds, 0 is returned.

**MMIOERR\_INVALID\_HANDLE**  
The handle passed was not correct.

**MMIOERR\_CANNOTWRITE**  
The I/O buffer needs to be written to disk but disk space is lacking.

**MMIOERR\_WRITE\_FAILED**  
Unable to write the buffer to disk. Possible hardware problem.

**MMIO\_WARNING**  
The file was closed, but the IOProc might be expecting additional data.

---

## mmioClose - Parameters

**hmmio** ([HMMIO](#)) - input  
The open file handle returned by [mmioOpen](#).

**usFlags** ([USHORT](#)) - input  
Contains nothing or the following flag:



MMIO\_FHOPEN

This flag is used to tell the I/O to not close the file or files of type FOURCC\_DOS. This allows an [HMMIO](#) instance to be closed in cases where a DOS file handle was provided to the I/O procedure on an [mmioOpen](#) call.

**rc** ([USHORT](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not correct.

MMIOERR\_CANNOTWRITE

The I/O buffer needs to be written to disk but disk space is lacking.

MMIOERR\_WRITE\_FAILED

Unable to write the buffer to disk. Possible hardware problem.

MMIO\_WARNING

The file was closed, but the IOProc might be expecting additional data.

-----

## mmioClose - Remarks

A buffer is automatically emptied when you close a file by calling mmioClose.

-----

## mmioClose - Related Functions

- [mmioOpen](#)
- [mmioRead](#)
- [mmioSeek](#)
- [mmioWrite](#)

-----

## mmioClose - Example Code

The following code illustrates how to close a file.

```
HMMIO hmmio1;
USHORT usFlags;
USHORT rc;
...

usFlags = 0;
rc = mmioClose(hmmio1, usFlags);
if (rc)
    /* error */
else
    ...
```

-----

## mmioClose - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

---

# mmioCreateChunk

---

## mmioCreateChunk - Syntax

This function creates a chunk in a RIFF file that was opened by [mmioOpen](#).

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO      hmmio;      /* Open file handle. */
PMMCKINFO  pckinfo;    /* Pointer to MMCKINFO. */
USHORT     usFlags;    /* Flags. */
USHORT     rc;         /* Return codes. */

rc = mmioCreateChunk(hmmio, pckinfo, usFlags);
```

---

## mmioCreateChunk Parameter - hmmio

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

---

## mmioCreateChunk Parameter - pckinfo

**pckinfo** ([PMMCKINFO](#)) - input

A pointer to an [MMCKINFO](#) data structure that is to be filled in as follows:

ckid

Must be the chunk ID of the chunk to create. If *usFlags* includes MMIO\_CREATERIFF or MMIO\_CREATELIST, this field will be filled in by mmioCreateChunk.

ckSize

Must be the size of the data portion of the chunk, including the form type or list type (if any) but not including the

8-byte chunk header or the terminating null (if any). If this value is not correct when [mmioAscend](#) is called to mark the end of the chunk, then [mmioAscend](#) will seek back and correct the chunk size.

fccType

Must contain the form type or list type, respectively, if *usFlags* contains MMIO\_CREATERIFF or MMIO\_CREATELIST.

ulDataOffset

This field will be filled in on the return from mmioCreateChunk. It will contain the file offset of the beginning of the data portion of the chunk.

ulFlags

This field will be filled in on the return from this mmioCreateChunk. It will contain the MMIO\_DIRTY flag to indicate this chunk was created with mmioCreateChunk.

-----

## mmioCreateChunk Parameter - usFlags

**usFlags** ([USHORT](#)) - input

Contains none or one of the following flags:

MMIO\_CREATERIFF

Create a chunk with an ID (*ckid* field) of RIFF and a form type in the *fccType* field.

MMIO\_CREATELIST

Create a chunk with an ID (*ckid* field) of LIST and a list type in the *fccType* field.

-----

## mmioCreateChunk Return Value - rc

**rc** ([USHORT](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

The parameter passed was not valid.

MMIOERR\_CANNOTWRITE

The I/O buffer needs to be written to disk but disk space is lacking.

-----

## mmioCreateChunk - Parameters

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

**pckinfo** ([PMMCKINFO](#)) - input

A pointer to an [MMCKINFO](#) data structure that is to be filled in as follows:

<code>ckid</code>	Must be the chunk ID of the chunk to create. If <i>usFlags</i> includes <code>MMIO_CREATERIFF</code> or <code>MMIO_CREATELIST</code> , this field will be filled in by <code>mmioCreateChunk</code> .
<code>ckSize</code>	Must be the size of the data portion of the chunk, including the form type or list type (if any) but not including the 8-byte chunk header or the terminating null (if any). If this value is not correct when <code>mmioAscend</code> is called to mark the end of the chunk, then <code>mmioAscend</code> will seek back and correct the chunk size.
<code>fccType</code>	Must contain the form type or list type, respectively, if <i>usFlags</i> contains <code>MMIO_CREATERIFF</code> or <code>MMIO_CREATELIST</code> .
<code>ulDataOffset</code>	This field will be filled in on the return from <code>mmioCreateChunk</code> . It will contain the file offset of the beginning of the data portion of the chunk.
<code>ulFlags</code>	This field will be filled in on the return from this <code>mmioCreateChunk</code> . It will contain the <code>MMIO_DIRTY</code> flag to indicate this chunk was created with <code>mmioCreateChunk</code> .
<b>usFlags (USHORT)</b> - input	
Contains none or one of the following flags:	
<code>MMIO_CREATERIFF</code>	Create a chunk with an ID ( <i>ckid</i> field) of RIFF and a form type in the <i>fccType</i> field.
<code>MMIO_CREATELIST</code>	Create a chunk with an ID ( <i>ckid</i> field) of LIST and a list type in the <i>fccType</i> field.
<b>rc (USHORT)</b> - returns	
Return codes indicating success or type of failure:	
<code>MMIO_SUCCESS</code>	If the function succeeds, 0 is returned.
<code>MMIOERR_INVALID_HANDLE</code>	The handle passed was not valid.
<code>MMIOERR_INVALID_PARAMETER</code>	The parameter passed was not valid.
<code>MMIOERR_CANNOTWRITE</code>	The I/O buffer needs to be written to disk but disk space is lacking.

-----

## mmioCreateChunk - Remarks

`mmioCreateChunk` creates a new chunk; that is, it writes a chunk header starting at the current file position and descends into the chunk. The chunk ID is copied from the *ckid* field of the provided `MMCKINFO` structure. Call `mmioAscend` after the chunk data has been written. The *ckSize* field is assumed to be a proposed chunk size, if it turns out to be correct; that is, if you write that much data into the chunk before calling `mmioAscend` to end the chunk, `mmioAscend` will not have to seek back and correct the chunk header.

-----

## mmioCreateChunk - Related Functions

- [mmioAscend](#)
- [mmioDescend](#)
- [mmioFOURCC](#)
- [mmioStringToFOURCC](#)

---

## mmioCreateChunk - Example Code

The following code illustrates how to create a chunk in a file.

```
HMMIO hmmiol;
MMCKINFO mmckinfo;
USHORT usFlags;
USHORT rc;
...

memset( &mmckinfo, '\0', sizeof(MMCKINFO) );
mmckinfo.ckid = FOURCC_WAVE;
mmckinfo.ckSize = 1000;
usFlags |= MMIO_CREATERIFF;

rc = mmioCreateChunk(hmmiol, &mmckinfo, usFlags);

if (rc)
    /* error */
else
    ...
```

---

## mmioCreateChunk - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioDescend

---

## mmioDescend - Syntax

This function descends into a chunk beginning at the current file position, or searches for a specified chunk.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO      hmmio;          /* Open file handle. */
PMMCKINFO  pckinfo;        /* Pointer to MMCKINFO. */
PMMCKINFO  pckinfoParent;  /* Pointer to MMCKINFO. */
USHORT     usFlags;        /* Flags. */
USHORT     rc;             /* Return codes. */
```

```
rc = mmioDescend(hmmio, pckinfo, pckinfoParent,
                 usFlags);
```

---

## mmioDescend Parameter - hmmio

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

---

## mmioDescend Parameter - pckinfo

**pckinfo** ([PMMCKINFO](#)) - input

A pointer to the caller-supplied [MMCKINFO](#) structure that is to be filled in as follows:

ckid	Set to the chunk ID of the chunk.
ckSize	Set to the size of the data portion of the chunk, including the form type or list type (if any) but not including the 8-byte chunk header or the terminating null (which is present only if chunk size is odd).
fccType	The form type for RIFF chunks, the list type for LIST types, or a NULL value.
ulDataOffset	The file offset of the beginning of the data portion of the chunk, which begins after the 8-byte chunk header. If the chunk is a LIST chunk or a RIFF chunk, then this field must contain the offset of the list type or form type.
ulFlags	Contains other information about the chunk. Currently, mmioDescend zeros this field.

---

## mmioDescend Parameter - pckinfoParent

**pckinfoParent** ([PMMCKINFO](#)) - input

Specifies a pointer to the [MMCKINFO](#) data structure, which is an optional caller-supplied structure that refers to the parent of the chunk that is being searched for.

A parent of a chunk is the enclosing chunk - only RIFF and LIST chunks can be parents. If *pckinfoParent* is given, it is assumed that *pckinfoParent* was filled in when mmioDescend was called to descend into the parent chunk, and mmioDescend will only search for and descend into a chunk within the parent chunk. If *pckinfoParent* is NULL, this restriction is not imposed. mmioDescend checks only if a chunk is past the end of a given parent chunk, not before the beginning of the parent chunk. Also, mmioDescend checks only if the beginning of a chunk is past the end of the parent chunk.

---

## mmioDescend Parameter - usFlags

**usFlags** ([USHORT](#)) - input

Contains 0 or one of the following flags. If none of these flags are specified, mmioDescend descends into the chunk that starts at the current file position.

#### MMIO\_FINDCHUNK

Search for a chunk with a specific ID. The *ckid* field of [MMCKINFO](#) passed in on the *pckinfo* parameter should contain the ID of the chunk to search for when *mmioDescend* is called.

#### MMIO\_FINDRIFF

Search for a chunk with an ID of FOURCC\_RIFF and with a specific form type. The *fccType* field of [MMCKINFO](#) passed in on the *pckinfo* parameter contains the form type of the RIFF chunk to search for when *mmioDescend* is called.

#### MMIO\_FINDLIST

Search for a chunk with an ID of FOURCC\_LIST and with a specific list type. The *fccType* field of [MMCKINFO](#) passed in on the *pckinfo* parameter contains the list type of the LIST chunk to search for when *mmioDescend* is called.

-----

## mmioDescend Return Value - rc

**rc** ([USHORT](#)) - returns

Return codes indicating success or type of failure:

#### MMIO\_SUCCESS

If the function succeeds, 0 is returned.

#### MMIOERR\_INVALID\_HANDLE

The handle passed was not correct.

#### MMIOERR\_INVALID\_PARAMETER

A parameter passed was not correct.

#### MMIOERR\_CHUNKNOTFOUND

The end of the file (or the end of the parent chunk, if given) is reached before the desired chunk is found.

-----

## mmioDescend - Parameters

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

**pckinfo** ([PMMCKINFO](#)) - input

A pointer to the caller-supplied [MMCKINFO](#) structure that is to be filled in as follows:

*ckid*

Set to the chunk ID of the chunk.

*ckSize*

Set to the size of the data portion of the chunk, including the form type or list type (if any) but not including the 8-byte chunk header or the terminating null (which is present only if chunk size is odd).

*fccType*

The form type for RIFF chunks, the list type for LIST types, or a NULL value.

*ulDataOffset*

The file offset of the beginning of the data portion of the chunk, which begins after the 8-byte chunk header. If the chunk is a LIST chunk or a RIFF chunk, then this field must contain the offset of the list type or form type.

*ulFlags*

Contains other information about the chunk. Currently, *mmioDescend* zeros this field.

**pckinfoParent** ([PMMCKINFO](#)) - input

Specifies a pointer to the [MMCKINFO](#) data structure, which is an optional caller-supplied structure that refers to the parent of the chunk that is being searched for.

A parent of a chunk is the enclosing chunk - only RIFF and LIST chunks can be parents. If *pckinfoParent* is given, it is assumed that

*pckinfoParent* was filled in when `mmioDescend` was called to descend into the parent chunk, and `mmioDescend` will only search for and descend into a chunk within the parent chunk. If *pckinfoParent* is NULL, this restriction is not imposed. `mmioDescend` checks only if a chunk is past the end of a given parent chunk, not before the beginning of the parent chunk. Also, `mmioDescend` checks only if the beginning of a chunk is past the end of the parent chunk.

**usFlags** ([USHORT](#)) - input

Contains 0 or one of the following flags. If none of these flags are specified, `mmioDescend` descends into the chunk that starts at the current file position.

**MMIO\_FINDCHUNK**

Search for a chunk with a specific ID. The *ckid* field of [MMCKINFO](#) passed in on the *pckinfo* parameter should contain the ID of the chunk to search for when `mmioDescend` is called.

**MMIO\_FINDRIFF**

Search for a chunk with an ID of `FOURCC_RIFF` and with a specific form type. The *fccType* field of [MMCKINFO](#) passed in on the *pckinfo* parameter contains the form type of the RIFF chunk to search for when `mmioDescend` is called.

**MMIO\_FINDLIST**

Search for a chunk with an ID of `FOURCC_LIST` and with a specific list type. The *fccType* field of [MMCKINFO](#) passed in on the *pckinfo* parameter contains the list type of the LIST chunk to search for when `mmioDescend` is called.

**rc** ([USHORT](#)) - returns

Return codes indicating success or type of failure:

**MMIO\_SUCCESS**

If the function succeeds, 0 is returned.

**MMIOERR\_INVALID\_HANDLE**

The handle passed was not correct.

**MMIOERR\_INVALID\_PARAMETER**

A parameter passed was not correct.

**MMIOERR\_CHUNKNOTFOUND**

The end of the file (or the end of the parent chunk, if given) is reached before the desired chunk is found.

-----

## mmioDescend - Remarks

A RIFF chunk consists of a four-byte chunk ID *ckid* (type `FOURCC`), followed by a four-byte chunk size, *ckSize* (type `ULONG`), followed by the data portion of the chunk, followed by a 0 pad byte if *ckSize* is odd. If *ckid* is `FOURCC_RIFF` or `FOURCC_LIST`, then the first four bytes of the data portion of the chunk are a form type or list type, respectively. *ckSize* is the size of the chunk data, not including *ckid* or *ckSize* or the pad byte (if any), but including the form type or list type (if present).

When `mmioDescend` is called, it assumes that the current file position is the beginning of a chunk header. If *pckinfoParent* is given, `mmioDescend` assumes that the current file position is within *pckinfoParent* (a RIFF to LIST chunk). If `mmioDescend` succeeds, the current file position will be either just after the form type or list type (12 bytes from the beginning of the chunk ID) if the chunk ID is `FOURCC_RIFF` or `FOURCC_LIST`, or the start of the data portion of the chunk (8 bytes from the beginning of the chunk ID).

For efficiency of RIFF I/O, it is recommended that the *hmmio* parameter be set up for buffered I/O. Note that the constants, `FOURCC_RIFF` and `FOURCC_LIST`, are defined to be the four-character codes, RIFF and LIST, respectively.

-----

## mmioDescend - Related Functions

- [mmioAscend](#)
- [mmioCreateChunk](#)
- [mmioFOURCC](#)
- [mmioStringToFOURCC](#)



---

## mmioDescend - Example Code

The following code illustrates how to descend into a chunk of a file.

```
HMMIO hmmiol;
MMCKINFO mmckinfo;
USHORT usFlags = 0;
USHORT rc;
...

memset( &mmckinfo, '\0', sizeof(MMCKINFO) );
usFlags |= MMIO_FINDRIFF;
mmckinfo.ckid = FOURCC_WAVE;

rc = mmioDescend(hmmiol, &mmckinfo, usFlags);
if (rc)
    /* error */
else
    ...
```

---

## mmioDescend - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioDetermineSSIOProc

---

## mmioDetermineSSIOProc - Syntax

This function determines the storage system of the media data object.

```
#define INCL_MMIOOS2
#include <os2.h>

PSZ          pszFileName;          /* Media object file name. */
PMMIOINFO    pmmioinfo;           /* Pointer to MMIOINFO. */
PFOURCC      pfccStorageSystem;    /* Pointer to the FOURCC. */
PSZ          pszParsedRemainder;    /* Pointer to the parsed file name. */
ULONG        rc;                   /* Return codes. */

rc = mmioDetermineSSIOProc(pszFileName, pmmioinfo,
    pfccStorageSystem, pszParsedRemainder);
```

-----

## mmioDetermineSSIOProc Parameter - pszFileName

**pszFileName** ([PSZ](#)) - input  
The file name of the media object. This parameter can be NULL.

-----

## mmioDetermineSSIOProc Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#)) - input  
A pointer to a [MMIOINFO](#) data structure that might contain additional data. Normally this is NULL, but is needed for compound-file elements that are not completely valid.

-----

## mmioDetermineSSIOProc Parameter - pfccStorageSystem

**pfccStorageSystem** ([PFOURCC](#)) - in/out  
Pointer to the FOURCC of the storage system that is returned when successfully completed.

-----

## mmioDetermineSSIOProc Parameter - pszParsedRemainder

**pszParsedRemainder** ([PSZ](#)) - in/out  
Pointer to the parsed file name that is returned when successfully completed.

-----

## mmioDetermineSSIOProc Return Value - rc

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure. For information about DOS File errors, see *ulErrorRet* in [MMIOINFO](#).

**MMIO\_SUCCESS**  
If the function succeeds, 0 is returned.

**MMIO\_ERROR**  
Unable to determine the FOURCC of the IOProc.

**MMIOERR\_INVALID\_PARAMETER**  
An invalid parameter was passed.

---

## mmioDetermineSSIOProc - Parameters

**pszFileName** ([PSZ](#)) - input

The file name of the media object. This parameter can be NULL.

**pmmioinfo** ([PMMIOINFO](#)) - input

A pointer to a [MMIOINFO](#) data structure that might contain additional data. Normally this is NULL, but is needed for compound-file elements that are not completely valid.

**pfccStorageSystem** ([PFOURCC](#)) - in/out

Pointer to the FOURCC of the storage system that is returned when successfully completed.

**pszParsedRemainder** ([PSZ](#)) - in/out

Pointer to the parsed file name that is returned when successfully completed.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure. For information about DOS File errors, see *ulErrorRet* in [MMIOINFO](#).

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

Unable to determine the FOURCC of the IOProc.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

---

## mmioDetermineSSIOProc - Remarks

mmioDetermineSSIOProc processes the [MMIOINFO](#) data first to check for a storage system I/O procedure specified in the *fccChildIOProc* field. If it is not NULL, the *fccChildIOProc* is returned as the storage system FOURCC. Otherwise, the file name is parsed for a separator character; if one is found, the extension is converted to the storage system FOURCC. In this case, mmioDetermineSSIOProc returns the parsed string that consists of those characters following the separator character. The name is parsed from left to right.

---

## mmioDetermineSSIOProc - Related Functions

- [mmioIdentifyStorageSystem](#)
  - [mmioIdentifyFile](#)
- 

## mmioDetermineSSIOProc - Example Code

The following code illustrates how to determine the storage system of a data object.

```
MMIOINFO mmioinfo;  
FOURCC fccType;  
PSZ pszParsedFileName;
```

```

ULONG rc;
...

rc = mmioDetermineSSIOProc( "DAN.BND+SUE.WAV",
                           &mmioinfo,
                           &fccType,
                           pszParsedFileName);

if (rc)
    /* error */
else
    ...

```

-----

## mmioDetermineSSIOProc - Topics

- Select an item:
- [Syntax](#)
  - [Parameters](#)
  - [Returns](#)
  - [Remarks](#)
  - [Example Code](#)
  - [Related Functions](#)
  - [Glossary](#)

-----

## mmioFindElement

-----

## mmioFindElement - Syntax

This function enumerates the entries of a compound file. mmioFindElement is a 32-bit function that is also provided as a 16-bit entry point.

```

#define INCL_MACHDR
#include <os2.h>

ULONG    ulCode;           /* Enumeration code. */
PSZ      pszElement;       /* Pointer to element name. */
ULONG    ulElementLen;     /* Length of element buffer. */
PSZ      pszFile;          /* Pointer to compound-file name. */
ULONG    ulReserved;       /* Reserved. */
ULONG    rc;               /* Return codes. */

rc = mmioFindElement(ulCode, pszElement, ulElementLen,
                    pszFile, ulReserved);

```

-----

## mmioFindElement Parameter - ulCode

**ulCode** ([ULONG](#)) - input

The following flags are used to control the find operation:

MMIO\_FE\_FINDFIRST

Find the first element in the specified compound file.

MMIO\_FE\_FINDNEXT

Find the next element in the specified compound file.

MMIO\_FE\_FINDELEMENT

Search for an element in the specified compound file. MMIO\_FE\_FINDELEMENT supersedes a MMIO\_FE\_FINDFIRST/MMIO\_FE\_FINDNEXT search on the same file.

MMIO\_FE\_FINDEND

Complete the search of a compound file. MMIO\_FE\_FINDEND is called after MMIO\_FE\_FINDELEMENT, MMIO\_FE\_FINDNEXT, or MMIO\_FE\_FINDFIRST.

-----

## mmioFindElement Parameter - pszElement

**pszElement** ([PSZ](#)) - in/out

Pointer to a compound-file element name.

-----

## mmioFindElement Parameter - ulElementLen

**ulElementLen** ([ULONG](#)) - input

Length of the buffer the *pszElement* points to.

-----

## mmioFindElement Parameter - pszFile

**pszFile** ([PSZ](#)) - input

Pointer to a RIFF compound-file name. This parameter should contain just the name of the compound file and not include the element name specification.

-----

## mmioFindElement Parameter - ulReserved

**ulReserved** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

-----

## mmioFindElement Return Value - rc

**rc** (**ULONG**) - returns

Return code indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_CF\_ENTRY\_NOT\_FOUND

The element cannot be found.

MMIOERR\_INVALID\_PARAMETER

If any parameter is missing.

ERROR\_INVALID\_PARAMETER

If *ulReserved* is not zero.

ERROR\_BUFFER\_OVERFLOW

Element name is longer than *ulElementLen*.

-----

## mmioFindElement - Parameters

**ulCode** (**ULONG**) - input

The following flags are used to control the find operation:

MMIO\_FE\_FINDFIRST

Find the first element in the specified compound file.

MMIO\_FE\_FINDNEXT

Find the next element in the specified compound file.

MMIO\_FE\_FINDELEMENT

Search for an element in the specified compound file. MMIO\_FE\_FINDELEMENT supersedes a MMIO\_FE\_FINDFIRST/MMIO\_FE\_FINDNEXT search on the same file.

MMIO\_FE\_FINDEND

Complete the search of a compound file. MMIO\_FE\_FINDEND is called after MMIO\_FE\_FINDELEMENT, MMIO\_FE\_FINDNEXT, or MMIO\_FE\_FINDFIRST.

**pszElement** (**PSZ**) - in/out

Pointer to a compound-file element name.

**ulElementLen** (**ULONG**) - input

Length of the buffer the *pszElement* points to.

**pszFile** (**PSZ**) - input

Pointer to a RIFF compound-file name. This parameter should contain just the name of the compound file and not include the element name specification.

**ulReserved** (**ULONG**) - input

Reserved for future use and must be set to zero.

**rc** (**ULONG**) - returns

Return code indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_CF\_ENTRY\_NOT\_FOUND

The element cannot be found.

MMIOERR\_INVALID\_PARAMETER

If any parameter is missing.

ERROR\_INVALID\_PARAMETER

If *ulReserved* is not zero.

ERROR\_BUFFER\_OVERFLOW

Element name is longer than *ulElementLen*.

---

## mmioFindElement - Remarks

The mmioFindElement function is a high-level interface to enumerate elements from a compound file.

For MMIO\_FE\_FINDFIRST and MMIO\_FE\_FINDNEXT, the *pszElement* parameter contains the name of an element in the specified compound file upon return. Only one MMIO\_FE\_FINDFIRST and MMIO\_FE\_FINDNEXT sequence is supported for a file at any one time. If an element is not found, MMIOERR\_CF\_ENTRY\_NOT\_FOUND is returned and the *pszElement* parameter is set to an empty string.

For MMIO\_FE\_FINDELEMENT, the element name specified in *pszElement* is searched for. If the name is found, a zero return code is returned. If the element is not found, then MMIOERR\_CF\_ENTRY\_NOT\_FOUND is returned and the *pszElement* field is set to an empty string.

MMIO\_FE\_FINDEND should be called after the search is complete. This flag indicates the compound file should be closed.

MMIO\_FE\_FINDFIRST opens the compound file on the first invocation, and the file remains open until MMIO\_FE\_FINDEND is called. The MMIO\_FE\_FINDEND flag must be sent after completing the search in order to close the file.

---

## mmioFindElement - Related Functions

- [mmioRemoveElement](#)

---

## mmioFindElement - Example Code

The following code illustrates enumeration of compound-file entries.

```
ULONG rc;
CHAR  szElement[CCHMAXPATH];
rc=mmioFindElement(MMIO_FE_FINDFIRST,szElement,CCHMAXPATH,"TEST.Bnd",0);
while(!rc) {
    /* Save current szElement */
    ...
    rc=mmioFindElement(MMIO_FE_FINDNEXT, szElement,CCHMAXPATH,"TEST.Bnd",0);
}
```

---

## mmioFindElement - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)  
[Glossary](#)

---

## mmioFlush

---

### mmioFlush - Syntax

This function writes the I/O buffer of a file to disk, if the I/O buffer was written into. It also empties the buffer if requested.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO    hmmio;    /* Open file handle. */
USHORT    usFlags; /* Flags. */
USHORT    rc;      /* Return codes. */

rc = mmioFlush(hmmio, usFlags);
```

---

### mmioFlush Parameter - hmmio

**hmmio** ([HMMIO](#)) - input  
The open file handle returned by [mmioOpen](#).

---

### mmioFlush Parameter - usFlags

**usFlags** ([USHORT](#)) - input  
Contains none or the following flag:

<a href="#">MMIO_EMPTYBUF</a>	Empties the I/O buffer. The allocated buffer is not dropped, but the calling <a href="#">mmioGetInfo</a> will reveal that the <i>pchNext</i> field of the <a href="#">MMIOINFO</a> structure will point to <i>pchEndRead</i> .
-------------------------------	--

---

### mmioFlush Return Value - rc

**rc** ([USHORT](#)) - returns  
Return codes indicating success or type of failure:



MMIO\_SUCCESS  
If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE  
The handle passed was not valid.

MMIOERR\_CANNOTWRITE  
The buffer could not be written to the disk. The disk could be full.

MMIOERR\_WRITE\_FAILED  
The buffer could not be written to the disk. This is a possible hardware problem.

MMIOERR\_NO\_BUFFER\_ALLOCATED  
A buffer was expected but was not present.

MMIOERR\_NO\_FLUSH\_NEEDED  
A mmioFlush function was requested, but the buffer was empty.

MMIOERR\_NO\_FLUSH\_FOR\_MEM\_FILE  
A mmioFlush was requested on a MEM file.

---

## mmioFlush - Parameters

**hmmio** (**HMMIO**) - input  
The open file handle returned by [mmioOpen](#).

**usFlags** (**USHORT**) - input  
Contains none or the following flag:

MMIO\_EMPTYBUF  
Empties the I/O buffer. The allocated buffer is not dropped, but the calling [mmioGetInfo](#) will reveal that the *pchNext* field of the **MMIOINFO** structure will point to *pchEndRead*.

**rc** (**USHORT**) - returns  
Return codes indicating success or type of failure:

MMIO\_SUCCESS  
If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE  
The handle passed was not valid.

MMIOERR\_CANNOTWRITE  
The buffer could not be written to the disk. The disk could be full.

MMIOERR\_WRITE\_FAILED  
The buffer could not be written to the disk. This is a possible hardware problem.

MMIOERR\_NO\_BUFFER\_ALLOCATED  
A buffer was expected but was not present.

MMIOERR\_NO\_FLUSH\_NEEDED  
A mmioFlush function was requested, but the buffer was empty.

MMIOERR\_NO\_FLUSH\_FOR\_MEM\_FILE  
A mmioFlush was requested on a MEM file.

---

## mmioFlush - Remarks

If the *hmmio* parameter represents a file that was opened using [mmioOpen](#), and *hmmio* parameter is currently set up for buffered I/O, and the buffer has been written into (by [mmioWrite](#), or by direct caller access to the buffer using [mmioGetInfo](#)) since the last time the buffer was flushed to disk, mmioFlush writes the buffer to the disk.

If the *hmmio* parameter is a memory file or is unbuffered, this function returns the appropriate error message indicated. Note that mmioFlush might fail if there is insufficient disk space to write the buffer, even if the preceding [mmioWrite](#) functions succeeded.

---

## mmioFlush - Related Functions

- [mmioSetBuffer](#)

---

## mmioFlush - Example Code

The following code illustrates how to write to disk.

```
HMMIO hmmiol;
USHORT usFlags = 0;
USHORT rc;
...

rc = mmioFlush( hmmiol, usFlags);
if (rc)
    /* error */
else
    ...
```

---

## mmioFlush - Topics

- Select an item:
- [Syntax](#)
  - [Parameters](#)
  - [Returns](#)
  - [Remarks](#)
  - [Example Code](#)
  - [Related Functions](#)
  - [Glossary](#)

---

## mmioFOURCC

---

## mmioFOURCC - Syntax

This macro converts four characters to a four-character code (FOURCC).

```
#define INCL_MMIOOS2
#include <os2.h>

CHAR      ch0; /* First character. */
CHAR      ch1; /* Second character. */
CHAR      ch2; /* Third character. */
CHAR      ch3; /* Fourth character. */
FOURCC    rc; /* Four-character code. */

rc = mmioFOURCC(ch0, ch1, ch2, ch3);
```

## mmioFOURCC Parameter - ch0

**ch0** (CHAR) - input  
The first character of the FOURCC code to be converted.

## mmioFOURCC Parameter - ch1

**ch1** (CHAR) - input  
The second character of the FOURCC code to be converted.

## mmioFOURCC Parameter - ch2

**ch2** (CHAR) - input  
The third character of the FOURCC code to be converted.

## mmioFOURCC Parameter - ch3

**ch3** (CHAR) - input  
The fourth character of the FOURCC code to be converted.

## mmioFOURCC Return Value - rc

**rc** ([FOURCC](#)) - returns

Returns the four-character code converted from the four characters as follows. Character *ch0* is copied to the lowest address and *ch3* is copied to the highest address.

-----

## mmioFOURCC - Parameters

**ch0** ([CHAR](#)) - input

The first character of the FOURCC code to be converted.

**ch1** ([CHAR](#)) - input

The second character of the FOURCC code to be converted.

**ch2** ([CHAR](#)) - input

The third character of the FOURCC code to be converted.

**ch3** ([CHAR](#)) - input

The fourth character of the FOURCC code to be converted.

**rc** ([FOURCC](#)) - returns

Returns the four-character code converted from the four characters as follows. Character *ch0* is copied to the lowest address and *ch3* is copied to the highest address.

-----

## mmioFOURCC - Remarks

This macro does not check to see if the four-character code follows any conventions regarding which characters to include in a FOURCC. The string is simply copied to a FOURCC and padded with blanks to the right, if required, or truncated to four characters, if required.

-----

## mmioFOURCC - Related Functions

- [mmioAscend](#)
- [mmioCreateChunk](#)
- [mmioDescend](#)
- [mmioStringToFOURCC](#)

-----

## mmioFOURCC - Example Code

The following code illustrates how to convert four characters to a four-character code.

```
#define FOURCC_WAVE = mmioFOURCC( 'W', 'A', 'V', 'E' );
```

-----

## mmioFOURCC - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

---

## mmioGetData

---

### mmioGetData - Syntax

This function allows an application to access the [MMIOINFO](#) structure of the file referenced by *hmmio* and can be used to call an I/O procedure directly.

Do *not* change any of the fields in the [MMIOINFO](#) structure as this information is stored within MMIO.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO      hmmio;      /* Open file handle. */
PMMIOINFO  pmmioinfo;  /* Information receiver. */
USHORT     usFlags;     /* Reserved. */
USHORT     rc;          /* Return codes. */

rc = mmioGetData(hmmio, pmmioinfo, usFlags);
```

---

### mmioGetData Parameter - hmmio

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

---

### mmioGetData Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#)) - in/out

A caller-allocated MMIOINFO buffer that is to receive information about the open file. See the description of the [mmioOpen](#) function for information about how the fields are interpreted.

---

# mmioGetData Parameter - usFlags

**usFlags** ([USHORT](#)) - input

Reserved for future use and must be set to zero.

---

## mmioGetData Return Value - rc

**rc** ([USHORT](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_UNBUFFERED

The specified file is not opened for buffered I/O.

MMIOERR\_READ\_FAILED

A read-advance operation failed.

MMIOERR\_SEEK\_FAILED

A seek operation prior to a write- or read-advance operation failed.

MMIOERR\_WRITE\_FAILED

A write-advance operation failed.

---

## mmioGetData - Parameters

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

**pmmioinfo** ([PMMIOINFO](#)) - in/out

A caller-allocated MMIOINFO buffer that is to receive information about the open file. See the description of the [mmioOpen](#) function for information about how the fields are interpreted.

**usFlags** ([USHORT](#)) - input

Reserved for future use and must be set to zero.

**rc** ([USHORT](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_UNBUFFERED

The specified file is not opened for buffered I/O.

MMIOERR\_READ\_FAILED

A read-advance operation failed.

MMIOERR\_SEEK\_FAILED

A seek operation prior to a write- or read-advance operation failed.

MMIOERR\_WRITE\_FAILED

A write-advance operation failed.

-----

## mmioGetData - Remarks

mmioGetData fills in all of the fields of the [MMIOINFO](#) structure, whereas [mmioGetInfo](#) fills in only the buffered I/O fields. Because an application requires a complete copy of the [MMIOINFO](#) structure when calling an I/O procedure directly, [mmioGetInfo](#) cannot be used.

-----

## mmioGetData - Related Functions

- [mmioGetInfo](#)

-----

## mmioGetData - Example Code

The following code illustrates how to access the MMIOINFO data structure.

```
HMMIO hmmiol;
MMIOINFO mmioinfo;
USHORT usFlags = 0;
USHORT rc;
...

memset( &mmioinfo, '\0', sizeof(MMIOINFO) );

rc = mmioGetData( hmmiol, &mmioinfo, usFlags);
if (rc)
    /* error */
else
    ...
```

-----

## mmioGetData - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioGetFormatName

---

## mmioGetFormatName - Syntax

This function provides the descriptive name of the format supported by the I/O procedure.

```
#define INCL_MMIOOS2
#include <os2.h>

PMMFORMATINFO    pmmformatinfo; /* Pointer to MMFORMATINFO data structure. */
PSZ               pszFormatName; /* Pointer to format name. */
PLONG             plBytesRead;    /* Pointer to a LONG. */
ULONG             ulReserved;    /* Reserved. */
ULONG             ulFlags;       /* Reserved. */
ULONG             rc;            /* Return codes. */

rc = mmioGetFormatName(pmmformatinfo, pszFormatName,
    plBytesRead, ulReserved, ulFlags);
```

---

## mmioGetFormatName Parameter - pmmformatinfo

**pmmformatinfo** ([PMMFORMATINFO](#)) - input

Pointer to an [MMFORMATINFO](#) structure that contains the *fccIOProc* field (FOURCC code of the IOProc) and the *lNameLength* field, (the length in bytes of the format pointed to by the *pszFormatName* parameter).

---

## mmioGetFormatName Parameter - pszFormatName

**pszFormatName** ([PSZ](#)) - output

Pointer to a format name. This function fills in the format name associated with the specified IOProc, up to the length, in bytes, specified by the *lNameLength* field. Make sure the buffer is *lNameLength* + 1 in length to handle the string terminator character.

---

## mmioGetFormatName Parameter - plBytesRead



**plBytesRead** ([PLONG](#)) - output  
Pointer to a LONG. The number of bytes read into *pszFormatName* is returned, representing the length of the format name.

## mmioGetFormatName Parameter - ulReserved

**ulReserved** ([ULONG](#)) - input  
Reserved for future use and must be set to zero.

## mmioGetFormatName Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input  
Reserved for future use and must be set to zero.

## mmioGetFormatName Return Value - rc

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure:

**MMIO\_SUCCESS**  
If the function succeeds, 0 is returned.

**MMIO\_ERROR**  
The function failed for a reason different from any of the following return.

**MMIOERR\_INVALID\_PARAMETER**  
An invalid parameter was passed.

## mmioGetFormatName - Parameters

**pmmformatinfo** ([PMMFORMATINFO](#)) - input  
Pointer to an [MMFORMATINFO](#) structure that contains the *fccIOProc* field (FOURCC code of the IOProc) and the *lNameLength* field, (the length in bytes of the format pointed to by the *pszFormatName* parameter).

**pszFormatName** ([PSZ](#)) - output  
Pointer to a format name. This function fills in the format name associated with the specified IOProc, up to the length, in bytes, specified by the *lNameLength* field. Make sure the buffer is *lNameLength* + 1 in length to handle the string terminator character.

**plBytesRead** ([PLONG](#)) - output  
Pointer to a LONG. The number of bytes read into *pszFormatName* is returned, representing the length of the format name.

**ulReserved** ([ULONG](#)) - input  
Reserved for future use and must be set to zero.

**ulFlags** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The function failed for a reason different from any of the following return.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

-----

## mmioGetFormatName - Remarks

An application can use this function in conjunction with the [mmioIdentifyFile](#) function to determine the size of the buffer needed to supply this call.

-----

## mmioGetFormatName - Related Functions

- [mmioGetFormats](#)
- [mmioIdentifyFile](#)

-----

## mmioGetFormatName - Example Code

The following code illustrates how to determine the format name associated with the IOProc.

```
MMFORMATINFO mmFormatInfo;
PSZ pszFormatName;
USHORT lBytesRead;
ULONG ulReserved = 0L;
ULONG ulFlags = 0L;
ULONG rc;
...

memset( &mmFormatInfo, '\\0', sizeof(MMFORMATINFO) );
mmFormatInfo.lNameLength = 40L;
mmFormatInfo.fccIOProc = FOURCC_BND;
mmFormatInfo.ulStructLen=sizeof(MMFORMATINFO);

rc = mmioGetFormatName( &mmformatinfo,
                        pszFormatName,
                        &lBytesRead,
                        ulReserved,
                        ulFlags);

if (rc)
    /* error */
else
    ...
```

-----

# mmioGetFormatName - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

---

## mmioGetFormats

---

## mmioGetFormats - Syntax

This function provides a list of all I/O procedures available for use.

```
#define INCL_MMIOOS2
#include <os2.h>

PMMFORMATINFO    pmmformatinfo;    /* Pointer to MMFORMATINFO structure. */
LONG              lNumFormats;      /* Number of formats. */
PVOID             pFormatInfoList;  /* Pointer to memory-allocated structure. */
PLONG             plFormatsRead;     /* Number of formats read. */
ULONG            ulReserved;        /* Reserved. */
ULONG            ulFlags;           /* Reserved. */
ULONG            rc;                /* Return codes. */

rc = mmioGetFormats(pmmformatinfo, lNumFormats,
    pFormatInfoList, plFormatsRead, ulReserved,
    ulFlags);
```

---

## mmioGetFormats Parameter - pmmformatinfo

**pmmformatinfo** (**PMMFORMATINFO**) - in/out

A pointer to an **MMFORMATINFO** structure that might have optionally set the *fcc/OProc* field (FOURCC code) or *ulMediaType* field (multimedia data type). These two fields provide the search criteria for matching an **MMFORMATINFO** structure. If both of these fields are NULL, then all I/O procedure **MMFORMATINFO** structures are returned, provided enough space is allocated for in the buffer pointed to by the *pFormatInfoList* parameter.

---

## mmioGetFormats Parameter - lNumFormats

**INumFormats** ([LONG](#)) - input

The maximum number of [MMFORMATINFO](#) structures that can be returned in the *pFormatInfoList* parameter buffer.

---

## mmioGetFormats Parameter - pFormatInfoList

**pFormatInfoList** ([PVOID](#)) - in/out

Pointer to a buffer that will be filled with a list of matched [MMFORMATINFO](#) structures. The application needs to allocate enough memory to hold the requested number of structures.

---

## mmioGetFormats Parameter - plFormatsRead

**plFormatsRead** ([PLONG](#)) - output

Pointer to a [LONG](#) data type. Returns the number of formats that were returned in the *pFormatInfoList* parameter buffer.

---

## mmioGetFormats Parameter - ulReserved

**ulReserved** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

---

## mmioGetFormats Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

---

## mmioGetFormats Return Value - rc

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

[MMIO\\_SUCCESS](#)

If the function succeeds, 0 is returned.

[MMIO\\_ERROR](#)

The function failed for a reason different from any of the following returns in this list.

[MMIOERR\\_INVALID\\_PARAMETER](#)

An invalid parameter was passed.

MMIOERR\_INTERNAL\_SYSTEM

An internal system error occurred.

-----

## mmioGetFormats - Parameters

**pmmformatinfo** ([PMMFORMATINFO](#)) - in/out

A pointer to an [MMFORMATINFO](#) structure that might have optionally set the *fccIOProc* field (FOURCC code) or *ulMediaType* field (multimedia data type). These two fields provide the search criteria for matching an [MMFORMATINFO](#) structure. If both of these fields are NULL, then all I/O procedure [MMFORMATINFO](#) structures are returned, provided enough space is allocated for in the buffer pointed to by the *pFormatInfoList* parameter.

**INumFormats** ([LONG](#)) - input

The maximum number of [MMFORMATINFO](#) structures that can be returned in the *pFormatInfoList* parameter buffer.

**pFormatInfoList** ([PVOID](#)) - in/out

Pointer to a buffer that will be filled with a list of matched [MMFORMATINFO](#) structures. The application needs to allocate enough memory to hold the requested number of structures.

**piFormatsRead** ([PLONG](#)) - output

Pointer to a LONG data type. Returns the number of formats that were returned in the *pFormatInfoList* parameter buffer.

**ulReserved** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

**ulFlags** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The function failed for a reason different from any of the following returns in this list.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_INTERNAL\_SYSTEM

An internal system error occurred.

-----

## mmioGetFormats - Remarks

An application can use the `mmioQueryFormatCount` function to query the number of formats supported. It can then call `mmioGetFormats` with the correct size of *pFormatInfoList* to obtain descriptive information about the file formats supported by currently installed I/O procedures. This listing will assist you in finding out which data types can be output to a device. You can also use `mmioGetFormats` to query the number of file formats supported. To allocate the buffer for the file formats supported, multiply the number of formats by the size of the [MMFORMATINFO](#) structure. (The [MMFORMATINFO](#) structures are all the same size.)

-----

## mmioGetFormats - Related Functions

- [mmioQueryFormatCount](#)
- [mmioGetFormatName](#)

---

## mmioGetFormats - Example Code

The following code illustrates how to obtain information about formats supported by currently installed I/O procedures.

```
MMFORMATINFO mmformatinfo;
LONG    lNumFormats;
PCHAR   pFormatInfoList;
LONG    lFormatsRead;
PCHAR   pFormatInfoList;
ULONG   ulReserved = 0L;
ULONG   ulFlags = 0L;
ULONG   rc;
...

memset( &mmformatinfo, '\0', sizeof(MMFORMATINFO) );
mmformatinfo.ulMediaType |= MMIO_MEDIATYPE_AUDIO;
lNumFormats = 3;

rc = mmioGetFormats( &mmformatinfo,
                    lNumFormats,
                    pFormatInfoList,
                    &lFormatsRead,
                    ulReserved,
                    ulFlags);

if (rc)
    /* error */
else
    ...
```

---

## mmioGetFormats - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioGetHeader

---

## mmioGetHeader - Syntax

This function requests a media-specific header for an open file. The specific header depends on the media type of the file and current track

setting, in the case of multiple tracks. This header can be a raw header or a translated header.

This function does not change the current file position. It is highly recommended that `mmioGetHeader` be called before performing [mmioRead](#) because this information is normally required to understand the data that is read.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO    hmmio;          /* Open file handle. */
PVOID    pHeader;        /* Pointer to header structure. */
LONG     lHeaderLength;  /* Header structure size. */
PLONG    plBytesRead;    /* # of bytes read. */
ULONG    ulReserved;     /* Reserved. */
ULONG    ulFlags;        /* Reserved. */
ULONG    rc;             /* Return codes. */

rc = mmioGetHeader(hmmio, pHeader, lHeaderLength,
                  plBytesRead, ulReserved, ulFlags);
```

---

## mmioGetHeader Parameter - hmmio

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

---

## mmioGetHeader Parameter - pHeader

**pHeader** ([PVOID](#)) - input

Pointer to a header structure. This structure is filled in by the IOProc. If the `MMIO_TRANSLATEHEADER` flag was set in the *ulTranslate* field of [MMIOINFO](#) on the [mmioOpen](#), then the header returned is one associated with the standard presentation format for that particular media type. Each media type has a different header.

The I/O procedure is expected to transpose native header information, as read from the file, into the standard presentation format header before passing the data to the caller. The currently defined values for each media type (*ulMediaType*) and their respective media structures are as follows:

**Note:** If `MMIO_NOTTRANSLATE` was specified on the open (default case) then the file format native header is returned.

`MMIO_MEDIATYPE_IMAGE`

The data represents a still image. Images use [MMIMAGEHEADER](#) as the media structure.

`MMIO_MEDIATYPE_AUDIO`

The data represents digital audio. Digital-audio data streams use [MMAUDIOHEADER](#) as the media structure.

`MMIO_MEDIATYPE_MIDI`

The data represents MIDI streams. MIDI data streams use [MMMIDIHEADER](#) as the media structure.

`MMIO_MEDIATYPE_DIGITALVIDEO`

The data represents digital video. Digital video data streams use [MMVIDEOHEADER](#) as the media structure.

`MMIO_MEDIATYPE_MOVIE`

The data represents a movie. Movie data uses [MMMIEHEADER](#) as the media structure.

---

## mmioGetHeader Parameter - lHeaderLength

**lHeaderLength** (**LONG**) - input  
The size, in bytes, of the header structure.

---

## mmioGetHeader Parameter - plBytesRead

**plBytesRead** (**PLONG**) - in/out  
Returns the number of bytes read to the header structure.

---

## mmioGetHeader Parameter - ulReserved

**ulReserved** (**ULONG**) - input  
Reserved for future use and must be set to zero.

---

## mmioGetHeader Parameter - ulFlags

**ulFlags** (**ULONG**) - input  
Reserved for future use and must be set to zero.

---

## mmioGetHeader Return Value - rc

**rc** (**ULONG**) - returns  
Return codes indicating success or type of failure:

- MMIO\_SUCCESS**  
If the function succeeds, 0 is returned.
- MMIO\_ERROR**  
The specified file is not a media-file format type.
- MMIOERR\_INVALID\_PARAMETER**  
An invalid parameter was passed.
- MMIOERR\_INTERNAL\_SYSTEM**  
An internal system error occurred.
- MMIOERR\_SEEK\_FAILED**  
A seek operation prior to a write- or read-advance operation failed.

---



# mmioGetHeader - Parameters

**hmmio** (**HMMIO**) - input

The open file handle returned by [mmioOpen](#).

**pHeader** (**PVOID**) - input

Pointer to a header structure. This structure is filled in by the IOProc. If the MMIO\_TRANSLATEHEADER flag was set in the *ulTranslate* field of **MMIOINFO** on the [mmioOpen](#), then the header returned is one associated with the standard presentation format for that particular media type. Each media type has a different header.

The I/O procedure is expected to transpose native header information, as read from the file, into the standard presentation format header before passing the data to the caller. The currently defined values for each media type (*ulMediaType*) and their respective media structures are as follows:

**Note:** If MMIO\_NOTTRANSLATE was specified on the open (default case) then the file format native header is returned.

MMIO\_MEDIATYPE\_IMAGE

The data represents a still image. Images use [MMIMAGEHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_AUDIO

The data represents digital audio. Digital-audio data streams use [MMAUDIOHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_MIDI

The data represents MIDI streams. MIDI data streams use [MMMIDIHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_DIGITALVIDEO

The data represents digital video. Digital video data streams use [MMVIDEOHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_MOVIE

The data represents a movie. Movie data uses [MMMIEHEADER](#) as the media structure.

**lHeaderLength** (**LONG**) - input

The size, in bytes, of the header structure.

**plBytesRead** (**PLONG**) - in/out

Returns the number of bytes read to the header structure.

**ulReserved** (**ULONG**) - input

Reserved for future use and must be set to zero.

**ulFlags** (**ULONG**) - input

Reserved for future use and must be set to zero.

**rc** (**ULONG**) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The specified file is not a media-file format type.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_INTERNAL\_SYSTEM

An internal system error occurred.

MMIOERR\_SEEK\_FAILED

A seek operation prior to a write- or read-advance operation failed.

-----

## mmioGetHeader - Remarks

The *pBytesRead* parameter value might differ from the actual number of bytes read from the file in the case of translations.

Compound files are not supported by the mmioGetHeader function. Only non-compound files and compound-file elements are supported.

This function can be used in conjunction with the [mmioSet](#) function to query specific track headers from a multiple track movie file.

If the length passed in was not large enough to hold the header, MMIOERR\_INVALID\_BUFFER\_LENGTH is set in *ulErrorRet*. If the header is in error, MMIOERR\_INVALID\_STRUCTURE is set in *ulErrorRet*.

---

## mmioGetHeader - Related Functions

- [mmioSetHeader](#)
- [mmioIdentifyFile](#)

---

## mmioGetHeader - Example Code

The following code illustrates how to return the header of a file.

```
HMMIO  hmmio1;
PCHAR  pHeader;
USHORT lHeaderLength;
USHORT LONG lBytesRead;
ULONG  ulReserved = 0L;
ULONG  ulFlags = 0L;
ULONG  rc;
...

rc = mmioGetHeader( hmmio1,
                   pHeader,
                   lHeaderLength,
                   &lBytesRead,
                   ulReserved,
                   ulFlags);

if (rc)
    /* error */
else
    ...
```

---

## mmioGetHeader - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioGetInfo

---

## mmioGetInfo - Syntax

This function gets information about a file that was opened with [mmioOpen](#). It also allows the caller to access the I/O buffer directly.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO      hmmio;      /* Open file handle. */
PMMIOINFO  pmmioinfo; /* Information receiver. */
USHORT     usFlags;    /* Reserved. */
USHORT     rc;         /* Return codes. */

rc = mmioGetInfo(hmmio, pmmioinfo, usFlags);
```

---

## mmioGetInfo Parameter - hmmio

**hmmio** ([HMMIO](#)) - input  
The open file handle returned by [mmioOpen](#).

---

## mmioGetInfo Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#)) - in/out  
A caller-allocated MMIOINFO buffer that is to receive information about the open file. See the description of the [mmioOpen](#) function for information about how the fields are interpreted.

---

## mmioGetInfo Parameter - usFlags

**usFlags** ([USHORT](#)) - input  
Reserved for future use and must be set to zero.

---

## mmioGetInfo Return Value - rc

**rc** ([USHORT](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_UNBUFFERED

The specified file is not opened for buffered I/O.

MMIOERR\_READ\_FAILED

A read-advance operation failed.

MMIOERR\_SEEK\_FAILED

A seek operation prior to a write- or read-advance operation failed.

MMIOERR\_WRITE\_FAILED

A write-advance operation failed.

-----

## mmioGetInfo - Parameters

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

**pmmioinfo** ([PMMIOINFO](#)) - in/out

A caller-allocated MMIOINFO buffer that is to receive information about the open file. See the description of the [mmioOpen](#) function for information about how the fields are interpreted.

**usFlags** ([USHORT](#)) - input

Reserved for future use and must be set to zero.

**rc** ([USHORT](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_UNBUFFERED

The specified file is not opened for buffered I/O.

MMIOERR\_READ\_FAILED

A read-advance operation failed.

MMIOERR\_SEEK\_FAILED

A seek operation prior to a write- or read-advance operation failed.

MMIOERR\_WRITE\_FAILED

A write-advance operation failed.

-----

## mmioGetInfo - Remarks

An application can access the I/O buffer directly, as follows:

Call `mmioGetInfo`. The `pchNext` field of the `MMIOINFO` structure is a pointer to the next byte that can be read from or written to.

To read directly from the buffer, the application reads from the location pointed to by `pchNext` up to (but not including) the location pointed to by the `pchEndRead` pointer.

To write directly to the buffer, the application writes to the location pointed to by `pchNext` up to (but not including) the location pointed to by the `pchEndWrite` pointer.

Once `pchNext` is modified, do not call any MMIO functions (except for `mmioAdvance`) until `mmioSetInfo` is called. In particular, do not call `mmioRead` and `mmioWrite`. Once `mmioSetInfo` is called, the caller must stop accessing the I/O buffer directly, and revert to using `mmioRead` and `mmioWrite` to read and write the file.

To read beyond `pchEndRead` or write beyond `pchEndWrite`, call `mmioAdvance` to read and write the contents of the next full buffer. `mmioAdvance` will adjust various fields in your `MMIOINFO` block, including `pchNext`, `pchEndRead`, and `pchEndWrite`.

Before calling `mmioAdvance` or `mmioSetInfo`, make sure you set the `MMIO_DIRTY` flag of the `ulFlags` field of the `MMIOINFO` structure passed in the `pmmioinfo` parameter if you have written to the buffer. Otherwise, the buffer contents will not get written to the disk.

The caller must not move `pchNext` backward. No fields other than `pchNext` and the `MMIO_DIRTY` flag of `ulFlags` are to be modified.

-----

## mmioGetInfo - Related Functions

- [mmioAdvance](#)
- [mmioGetData](#)
- [mmioSetInfo](#)

-----

## mmioGetInfo - Example Code

The following code illustrates how to get information about a file.

```
HMMIO hmmiol;
MMIOINFO mmioinfo;
USHORT usFlags = 0;
USHORT rc;
...

memset( &mmioinfo, '\\0', sizeof(MMIOINFO) );

rc = mmioGetInfo( hmmiol, &mmioinfo, usFlags);
if (rc)
    /* error */
else
    ...
```

-----

## mmioGetInfo - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

# mmioGetLastError

---

## mmioGetLastError - Syntax

This function returns the last error condition stored in the *ulErrorRet* field of the [MMIOINFO](#) structure that might contain additional information for the analysis of the last error routine.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO    hmmio; /* Open file handle. */
ULONG    rc;     /* Return code. */

rc = mmioGetLastError(hmmio);
```

---

## mmioGetLastError Parameter - hmmio

**hmmio** ([HMMIO](#)) - input  
The open file handle returned by [mmioOpen](#).

---

## mmioGetLastError Return Value - rc

**rc** ([ULONG](#)) - returns  
Return code indicating success or type of failure:

MMIOERR_INVALID_HANDLE	An invalid handle was passed.
------------------------	-------------------------------

---

## mmioGetLastError - Parameters

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

**rc** ([ULONG](#)) - returns

Return code indicating success or type of failure:

MMIOERR\_INVALID\_HANDLE

An invalid handle was passed.

---

## mmioGetLastError - Remarks

The user can call mmioGetLastError for those functions that return only MMIO\_ERROR or MMIO\_CF\_FAILURE, and obtain additional information about the failing condition from the *ulErrorRet* field of the [MMIOINFO](#) structure. If *ulErrorRet* does not contain an MMIO error code, it contains an OS/2 error code or 0.

---

## mmioGetLastError - Example Code

The following code illustrates how to return the last error condition.

```
HMMIO hmmiol;  
ULONG rc;  
...  
  
rc = mmioGetLastError( hmmiol );  
if (rc)  
    /* error */  
else  
    ...
```

---

## mmioGetLastError - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Glossary](#)

---

## mmioidentifyFile

---

## mmioidentifyFile - Syntax

This function will determine (if possible) the format of a file by either using the file name or querying currently installed I/O procedures to see which I/O procedure can understand and process the specified file.

```
#define INCL_MMIOOS2
#include <os2.h>

PSZ      pszFileName;      /* Filename. */
PMMIOINFO pmmioinfo;      /* Pointer to MMIOINFO. */
PMMFORMATINFO pmmformatinfo; /* Pointer to MMFORMATINFO. */
PFOURCC   pfccStorageSystem; /* FOURCC pointer. */
ULONG     ulReserved;      /* Reserved. */
ULONG     ulFlags;         /* Flags. */
ULONG     rc;              /* Return codes. */

rc = mmioIdentifyFile(pszFileName, pmmioinfo,
    pmmformatinfo, pfccStorageSystem, ulReserved,
    ulFlags);
```

---

## mmioIdentifyFile Parameter - pszFileName

**pszFileName** ([PSZ](#)) - input  
The name of the file to identify.

---

## mmioIdentifyFile Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#)) - input  
Pointer to an [MMIOINFO](#) structure. This parameter is needed when a RIFF compound-file element is not completely valid. Normally this is NULL.

---

## mmioIdentifyFile Parameter - pmmformatinfo

**pmmformatinfo** ([PMMFORMATINFO](#)) - in/out  
Pointer to an [MMFORMATINFO](#) structure that, upon return from the function, contains information about the I/O procedure that handles this format. This includes the media type, such as image, audio, and compound, and the I/O procedure FOURCC code value that can then be used to open the file for further processing. This is returned only upon successful completion.

---

## mmioIdentifyFile Parameter - pfccStorageSystem

**pfccStorageSystem** ([PFOURCC](#)) - output  
A pointer that, upon return from the function, contains the FOURCC code of the storage system.



---

## mmioIdentifyFile Parameter - ulReserved

**ulReserved** ([ULONG](#)) - input  
Reserved for future use and must be set to zero.

---

## mmioIdentifyFile Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input  
The following flags are defined:

**MMIO\_FORCE\_IDENTIFY\_SS**  
Forces the identification of a storage system by ignoring the file name and actually checking the MMIO Manager's I/O procedure list.

**MMIO\_FORCE\_IDENTIFY\_FF**  
Forces the identification of a file format by ignoring the file name and actually checking the MMIO Manager's I/O procedure list.

---

## mmioIdentifyFile Return Value - rc

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure. For information about DOS File errors, use the [mmioGetLastError](#) function.

**MMIO\_SUCCESS**  
If the function succeeds, 0 is returned.

**MMIO\_ERROR**  
The function failed for a reason different from any of the following returns in this list.

**MMIOERR\_INVALID\_PARAMETER**  
An invalid parameter was passed.

**MMIOERR\_INTERNAL\_SYSTEM**  
An internal system error occurred.

---

## mmioIdentifyFile - Parameters

**pszFileName** ([PSZ](#)) - input  
The name of the file to identify.

**pmmioinfo** ([PMMIOINFO](#)) - input  
Pointer to an [MMIOINFO](#) structure. This parameter is needed when a RIFF compound-file element is not completely valid. Normally this is NULL.

**pmmformatinfo** ([PMMFORMATINFO](#)) - in/out

Pointer to an [MMFORMATINFO](#) structure that, upon return from the function, contains information about the I/O procedure that handles this format. This includes the media type, such as image, audio, and compound, and the I/O procedure FOURCC code value that can then be used to open the file for further processing. This is returned only upon successful completion.

**pfccStorageSystem** ([PFOURCC](#)) - output

A pointer that, upon return from the function, contains the FOURCC code of the storage system.

**ulReserved** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

**ulFlags** ([ULONG](#)) - input

The following flags are defined:

**MMIO\_FORCE\_IDENTIFY\_SS**

Forces the identification of a storage system by ignoring the file name and actually checking the MMIO Manager's I/O procedure list.

**MMIO\_FORCE\_IDENTIFY\_FF**

Forces the identification of a file format by ignoring the file name and actually checking the MMIO Manager's I/O procedure list.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure. For information about DOS File errors, use the [mmioGetLastError](#) function.

**MMIO\_SUCCESS**

If the function succeeds, 0 is returned.

**MMIO\_ERROR**

The function failed for a reason different from any of the following returns in this list.

**MMIOERR\_INVALID\_PARAMETER**

An invalid parameter was passed.

**MMIOERR\_INTERNAL\_SYSTEM**

An internal system error occurred.

---

## mmioIdentifyFile - Remarks

The order of I/O procedures to be searched is controlled by the MMPMMMIO.INI file; the order in which they are installed is controlled by [mmioInstallIOProc](#). The last installed procedure is first in the list. You can control the order of search based on the formats you normally use.

The default match will be the DOS I/O procedure.

---

## mmioIdentifyFile - Related Functions

- [mmioOpen](#)
- [mmioIdentifyStorageSystem](#)
- [mmioDetermineSSIOProc](#)

---

## mmioIdentifyFile - Example Code

The following code illustrates how to determine the file processing capability of the IOProcs.

```
MMIOINFO mmioinfo;
```



## mmioIdentifyStorageSystem Parameter - pszFileName

**pszFileName** ([PSZ](#)) - input  
The file name to be identified.

-----

## mmioIdentifyStorageSystem Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#)) - input  
A pointer to the [MMIOINFO](#) buffer that might contain additional data. Normally this is NULL, but is needed for compound-file elements when they are not fully qualified.

-----

## mmioIdentifyStorageSystem Parameter - pfccStorageSystem

**pfccStorageSystem** ([PFOURCC](#)) - output  
Pointer to the FOURCC code of the storage system that gets returned upon successful completion.

-----

## mmioIdentifyStorageSystem Return Value - rc

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure. For more information about DOS File errors, use the [mmioGetLastError](#) function.

MMIO\_SUCCESS  
If the function succeeds, 0 is returned.

MMIO\_ERROR  
The specified file is not a storage-system type.

MMIOERR\_INVALID\_PARAMETER  
An invalid parameter was passed.

MMIOERR\_INTERNAL\_SYSTEM  
An internal system error occurred.

-----

## mmioIdentifyStorageSystem - Parameters

**pszFileName** ([PSZ](#)) - input  
The file name to be identified.

**pmmioinfo** ([PMMIOINFO](#)) - input

A pointer to the [MMIOINFO](#) buffer that might contain additional data. Normally this is NULL, but is needed for compound-file elements when they are not fully qualified.

**pfccStorageSystem** ([PFOURCC](#)) - output

Pointer to the FOURCC code of the storage system that gets returned upon successful completion.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure. For more information about DOS File errors, use the [mmioGetLastError](#) function.

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The specified file is not a storage-system type.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_INTERNAL\_SYSTEM

An internal system error occurred.

-----

## mmioIdentifyStorageSystem - Remarks

mmioIdentifyStorageSystem processes the MMIO internal I/O procedure list to determine if the file name specified is of type MMIO\_IOPROC\_STORAGESYSTEM. If it is, an MMIOM\_IDENTIFYFILE message is sent to the I/O procedure to see if it can identify the data object. The *pfccStorageSystem* parameter contains the FOURCC code of the I/O procedure that successfully identified the data object.

-----

## mmioIdentifyStorageSystem - Related Functions

- [mmioDetermineSSIOProc](#)
- [mmioIdentifyFile](#)

-----

## mmioIdentifyStorageSystem - Example Code

The following code illustrates how to determine the storage system.

```
PSZ pszFileName;
MMIOINFO mmioinfo;
FOURCC fccStorageSystem;
ULONG rc;
...

memset( &mmioinfo, '\0', sizeof(MMIOINFO) );
mmioinfo.fccIOProc = FOURCC_BND;
strcpy( pszFileName, "myfile.bnd+element.foo" );

rc = mmioIdentifyStorageSystem( pszFileName,
                                &mmioinfo,
                                &fccStorageSystem);

if (rc)
    /* error */
else
{
    if (!fccStorageSystem)
    {
```

```

        return (MMIO_ERROR);
    }
    else
    {
        mmioinfo.fccChildIOProc = fccStorageSystem;
    }
}
...

```

---

## mmioIdentifyStorageSystem - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioIniFileCODEC

---

## mmioIniFileCODEC - Syntax

This function modifies the initialization file (MMPMMMIO.INI) for MMIO Manager services. It adds, replaces, removes, or finds a CODEC entry in the MMPMMMIO.INI file.

```

#define INCL_MMIOOS2
#define INCL_MMIO_CODEC
#include <os2.h>

PCODECINIFILEINFO  pCODECIniFileInfo; /* Pointer. */
ULONG              ulFlags;           /* Flags. */
ULONG              rc;                 /* Return codes. */

rc = mmioIniFileCODEC(pCODECIniFileInfo, ulFlags);

```

---

## mmioIniFileCODEC Parameter - pCODECIniFileInfo

**pCODECIniFileInfo** ([PCODECINIFILEINFO](#)) - in/out

A pointer to the [CODECINIFILEINFO](#) structure that contains the file format FOURCC, compression type, compression subtype, CODEC DLL name, entry procedure name, and other CODEC Procedure information.

---

## mmioIniFileCODEC Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input

Specifies options for the operation. Contains one or more of the following flags:

**MMIO\_INSTALLPROC**

Adds a CODEC Proc to the end of the MMPMMMIO.INI file. If an existing entry in the table matches the new entry, the new entry replaces the existing entry. An entry match is determined by specifying 0 or more of the match flags. If none are specified, the default is to match on the FOURCC.

**MMIO\_REMOVEPROC**

Deletes a matching entry from the MMPMMMIO.INI file. An entry match is determined by specifying 0 or more of the match flags. If none are specified, the default is to match on the FOURCC.

**MMIO\_FINDPROC**

Finds a matching entry from the MMPMMMIO.INI file. This fills in the remainder of the [CODECINIFILEINFO](#). An entry match is determined by specifying 0 or more of the match flags. If none are specified, the default is to match on the FOURCC.

**Note:** If MMIO\_MATCHFIRST is set, then MMIO\_FINDPROC does not default to the FOURCC.

**MMIO\_MATCHFIRST**

Finds the first entry in the MMPMMMIO.INI file if no match flags are specified. Otherwise, it finds the first entry that matches the contents of the fields specified by the match flags. In either case, the [CODECINIFILEINFO](#) structure is returned in the *pCODECIniFileInfo* parameter.

**MMIO\_MATCHNEXT**

If no match flags are specified, finds the next CODEC entry in the MMPMMMIO.INI file following the entry passed in *pCODECIniFileInfo*. If match flags are specified, finds the next entry that matches the search criteria specified by the flags. In either case, the *pCODECIniFileInfo* structure is returned.

**MMIO\_MATCHCOMPRESSTYPE**

Uses compression type (*ulCompressType* field of [CODECINIFILEINFO](#)) as a search criteria.

**MMIO\_MATCHCOMPRESSSUBTYPE**

Uses compression subtype (*ulCompressSubType* field of [CODECINIFILEINFO](#)) as a search criteria.

**MMIO\_MATCHHWID**

Uses hardware ID (*szHWID* field of [CODECINIFILEINFO](#)) as a search criteria.

**MMIO\_MATCHCAPSFLAGS**

Uses capability flags (*ulCapsFlags* of [CODECINIFILEINFO](#)) as a search criteria. Note that this search is not based on the exact match. If the target entry contains the flags, the match is satisfied.

**MMIO\_MATCHFOURCC**

Uses the FOURCC code (*fcc* field of [CODECINIFILEINFO](#)) as a search criteria.

**MMIO\_MATCHDLL**

Uses the DLL Name (*szDLLName* field of [CODECINIFILEINFO](#)) as a search criteria.

**MMIO\_MATCHPROCEDURENAME**

Uses the case-sensitive Procedure Name (*szProcName* field of [CODECINIFILEINFO](#)) as a search criteria.

**MMIO\_FULLPATH**

Uses the drive or path given with the DLL name (*szDLLName* field of [CODECINIFILEINFO](#)), otherwise use only the base file name. This allows DLLs with the same base name to be loaded from different directories.

---

## mmioIniFileCODEC Return Value - rc

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

**MMIO\_SUCCESS**

If the function succeeds, 0 is returned.

**MMIO\_ERROR**

The function failed for a reason different from any of the following returns in this list.

**MMIO\_INVALID\_PARAMETER**

An invalid parameter was passed.

**MMIOERR\_INTERNAL\_SYSTEM**

An internal system error was found.

**MMIOERR\_NO\_CORE**

Unable to allocate enough memory for the MMPMMMIO.INI data.

**MMIOERR\_INI\_OPEN**

Unable to open the MMPMMMIO.INI file.

**MMIOERR\_INVALID\_DLLNAME**

Unable to validate the DLL name.

**MMIOERR\_INVALID\_PROCEDURENAME**

Unable to validate the CODEC procedure name.

**MMIOERR\_MATCH\_NOT\_FOUND**

Unable to FIND the FOURCC, compression, DLL, or procedure name in MMPMMMIO.INI.

**MMIOERR\_CODEEC\_NOT\_SUPPORTED**

Although the file format is supported, the particular CODEC is not.

-----

## mmioIniFileCODEC - Parameters

**pCODECIniFileInfo** ([PCODECINIFILEINFO](#)) - in/out

A pointer to the [CODECINIFILEINFO](#) structure that contains the file format FOURCC, compression type, compression subtype, CODEC DLL name, entry procedure name, and other CODEC Procedure information.

**ulFlags** ([ULONG](#)) - input

Specifies options for the operation. Contains one or more of the following flags:

**MMIO\_INSTALLPROC**

Adds a CODEC Proc to the end of the MMPMMMIO.INI file. If an existing entry in the table matches the new entry, the new entry replaces the existing entry. An entry match is determined by specifying 0 or more of the match flags. If none are specified, the default is to match on the FOURCC.

**MMIO\_REMOVEPROC**

Deletes a matching entry from the MMPMMMIO.INI file. An entry match is determined by specifying 0 or more of the match flags. If none are specified, the default is to match on the FOURCC.

**MMIO\_FINDPROC**

Finds a matching entry from the MMPMMMIO.INI file. This fills in the remainder of the [CODECINIFILEINFO](#). An entry match is determined by specifying 0 or more of the match flags. If none are specified, the default is to match on the FOURCC.

**Note:** If MMIO\_MATCHFIRST is set, then MMIO\_FINDPROC does not default to the FOURCC.

**MMIO\_MATCHFIRST**

Finds the first entry in the MMPMMMIO.INI file if no match flags are specified. Otherwise, it finds the first entry that matches the contents of the fields specified by the match flags. In either case, the [CODECINIFILEINFO](#) structure is returned in the *pCODECIniFileInfo* parameter.

**MMIO\_MATCHNEXT**

If no match flags are specified, finds the next CODEC entry in the MMPMMMIO.INI file following the entry



passed in *pCODECIniFileInfo*. If match flags are specified, finds the next entry that matches the search criteria specified by the flags. In either case, the *pCODECIniFileInfo* structure is returned.

**MMIO\_MATCHCOMPRESSTYPE**

Uses compression type (*ulCompressType* field of [CODECINIFILEINFO](#)) as a search criteria.

**MMIO\_MATCHCOMPRESSSUBTYPE**

Uses compression subtype (*ulCompressSubType* field of [CODECINIFILEINFO](#)) as a search criteria.

**MMIO\_MATCHHHWID**

Uses hardware ID (*szHWID* field of [CODECINIFILEINFO](#)) as a search criteria.

**MMIO\_MATCHCAPSFLAGS**

Uses capability flags (*ulCapsFlags* of [CODECINIFILEINFO](#)) as a search criteria. Note that this search is not based on the exact match. If the target entry contains the flags, the match is satisfied.

**MMIO\_MATCHFOURCC**

Uses the FOURCC code (*fcc* field of [CODECINIFILEINFO](#)) as a search criteria.

**MMIO\_MATCHDLL**

Uses the DLL Name (*szDLLName* field of [CODECINIFILEINFO](#)) as a search criteria.

**MMIO\_MATCHPROCEDURENAME**

Uses the case-sensitive Procedure Name (*szProcName* field of [CODECINIFILEINFO](#)) as a search criteria.

**MMIO\_FULLPATH**

Uses the drive or path given with the DLL name (*szDLLName* field of [CODECINIFILEINFO](#)), otherwise use only the base file name. This allows DLLs with the same base name to be loaded from different directories.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

**MMIO\_SUCCESS**

If the function succeeds, 0 is returned.

**MMIO\_ERROR**

The function failed for a reason different from any of the following returns in this list.

**MMIO\_INVALID\_PARAMETER**

An invalid parameter was passed.

**MMIOERR\_INTERNAL\_SYSTEM**

An internal system error was found.

**MMIOERR\_NO\_CORE**

Unable to allocate enough memory for the MMPMMMIO.INI data.

**MMIOERR\_INI\_OPEN**

Unable to open the MMPMMMIO.INI file.

**MMIOERR\_INVALID\_DLLNAME**

Unable to validate the DLL name.

**MMIOERR\_INVALID\_PROCEDURENAME**

Unable to validate the CODEC procedure name.

**MMIOERR\_MATCH\_NOT\_FOUND**

Unable to FIND the FOURCC, compression, DLL, or procedure name in MMPMMMIO.INI.

**MMIOERR\_CODEC\_NOT\_SUPPORTED**

Although the file format is supported, the particular CODEC is not.

-----

## mmioIniFileCODEC - Remarks

The MMPMMMIO.INI file is in the directory specified in the MMBASE environment variable.

The DLL name (*szDLLName*) specified in the [CODECINIFILEINFO](#) structure must follow the same naming conventions as the DosLoadModule function. If the DLL or procedure name is invalid, an error is returned.

In a deletion, the entry is removed and the entire file is rewritten, to prevent it from growing with deleted entries. This is due to how OS/2 functions delete entries from INI files. Deleted entries are not reused.

## mmioIniFileCODEC - Example Code

The following code illustrates how to add, replace, remove, or find a CODEC entry in the MMPMMMIO.INI file.

```
CODECINIFILEINFO codecIniFileInfo;
ULONG ulFlags = 0L;
ULONG rc;
...

memset( &codecIniFileInfo, '\0', sizeof(CODECINIFILEINFO) );
codecIniFileInfo.ulStructLen = sizeof (CODECINIFILEINFO);
codecIniFileInfo.fcc = FOURCC_MYPROC;
codecIniFileInfo.ulCompressType = COMPRESSTYPE_MYPROC;
codecIniFileInfo.ulCompressSubType = COMPRESSSUBTYPE_MYPROC;
codecIniFileInfo.ulMediaType = MEDIATYPE_MYPROC;
codecIniFileInfo.ulCapsFlags = CODEC_DECOMPRESS;
codecIniFileInfo.szHWID = HWID_MYPROC;
codecIniFileInfo.ulMaxScrBuflen = MAXBUFLen_MYPROC;
codecIniFileInfo.ulSyncMethod = SYNCMETHOD_MYPROC;
codecIniFileInfo.ulXalignment = XALIGNMENT_MYPROC;
codecIniFileInfo.ulYalignment = YALIGNMENT_MYPROC;
strncpy( codecIniFileInfo.szDLLName, "MYPROC", DLLNAME_SIZE );
strncpy( codecIniFileInfo.szProcName, "MyCODECProc", PROCNAME_SIZE );
ulFlags = MMIO_INSTALLPROC
          MMIO_MATCHCOMPRESSTYPE | MMIO_MATCHCOMPRESSSUBTYPE;
          MMIO_MATCHCAPSFLAGS | MMIO_MATCHHWID
rc = mmioIniFileCODEC( &codecIniFileInfo,
                      ulFlags);

if (rc)
    /* error */
else
    ...
```

## mmioIniFileCODEC - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Example Code](#)
- [Glossary](#)

## mmioIniFileHandler

# mmioIniFileHandler - Syntax

This function adds, replaces, removes, or finds an I/O procedure in the initialization file (MMPMMMIO.INI) for MMIO services.

```
#define INCL_MMIOOS2
#include <os2.h>

PMMINIFILEINFO    pmmminifileinfo; /* Pointer to MMINIFILEINFO. */
ULONG             ulFlags;          /* Operation options. */
ULONG             rc;                /* Return codes. */

rc = mmioIniFileHandler(pmmminifileinfo, ulFlags);
```

---

## mmioIniFileHandler Parameter - pmmminifileinfo

**pmmminifileinfo** (**PMMINIFILEINFO**) - input

A pointer to the **MMINIFILEINFO** structure that contains the FOURCC code, DLL name, and the name of the I/O procedure.

---

## mmioIniFileHandler Parameter - ulFlags

**ulFlags** (**ULONG**) - input

Specifies options for the operation. Contains one of the following flags:

**MMIO\_INSTALLPROC**

Adds an I/O procedure to the end of the MMPMMMIO.INI file. If an existing entry in the table matches the new entry, the new entry replaces the existing entry. An entry match is determined by specifying 0 or more of the match flags. If none are specified, the default is to match on the FOURCC code.

**MMIO\_REMOVEPROC**

Deletes a matching entry from the MMPMMMIO.INI file. An entry match is determined by specifying 0 or more of the match flags. If none are specified, the default is to match on the FOURCC code.

**MMIO\_FINDPROC**

Finds a matching entry from the MMPMMMIO.INI file. This fills in the remainder of the **MMINIFILEINFO** structure passed in the *pmmminifileinfo* parameter. An entry match is determined by specifying 0 or more of the match flags. If none are specified, the default is to match on the FOURCC.

**Note:** If **MMIO\_FINDFIRST** is set, then **MMIO\_FINDPROC** does not default to the FOURCC.

**MMIO\_MATCHFIRST**

Finds the first entry in the MMPMMMIO.INI file if no match flags are specified. Otherwise, it finds the first entry that matches the contents of the fields specified by the match flags. In either case, the **MMINIFILEINFO** structure is returned.

**MMIO\_MATCHNEXT**

If no match flags are specified, this finds the next entry in the MMPMMMIO.INI file following the entry (**MMINIFILEINFO** structure) passed in the *pmmminifileinfo* parameter. If match flags are specified, this finds the next entry that matches the search criteria specified by the flags. In either case, the **MMINIFILEINFO** structure is returned.

**MMIO\_MATCHFOURCC**

Use the FOURCC code (*fccOProc* field of the **MMINIFILEINFO** structure) as a search criteria.

**MMIO\_MATCHDLL**

Use the DLL Name (*szDLLName* field of the [MMINIFILEINFO](#) structure) as a search criteria.

**MMIO\_MATCHPROCEDURENAME**

Use the case-sensitive procedure name (*szProcName* field of the [MMINIFILEINFO](#) structure) as a search criteria.

**MMIO\_FULLPATH**

Use the drive or path given with the DLL name (*szDLLName* field of the [MMINIFILEINFO](#) structure), otherwise use only the base file name. This allows DLLs with the same base name to be loaded from different directories.

**MMIO\_EXTENDED\_STRUCT**

This flag *must* be set for release 1.1 and later releases. It indicates the expanded structure is being used.

-----

## mmioIniFileHandler Return Value - rc

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

**MMIO\_SUCCESS**

If the function succeeds, 0 is returned.

**MMIO\_ERROR**

The function failed for a reason different from any of the following returns in this list.

**MMIOERR\_INVALID\_PARAMETER**

An invalid parameter was passed.

**MMIOERR\_INTERNAL\_SYSTEM**

An internal system error was found.

**MMIOERR\_NO\_CORE**

Unable to allocate enough memory for the MMPMMMIO.INI data.

**MMIOERR\_INI\_OPEN**

Unable to open the MMPMMMIO.INI file.

**MMIOERR\_INVALID\_DLLNAME**

Unable to validate the DLL name.

**MMIOERR\_INVALID\_PROCEDURENAME**

Unable to validate the procedure name.

**MMIOERR\_MATCH\_NOT\_FOUND**

Unable to find the I/O procedure to satisfy the input criteria.

-----

## mmioIniFileHandler - Parameters

**pmmminifileinfo** ([PMMINIFILEINFO](#)) - input

A pointer to the [MMINIFILEINFO](#) structure that contains the FOURCC code, DLL name, and the name of the I/O procedure.

**ulFlags** ([ULONG](#)) - input

Specifies options for the operation. Contains one of the following flags:

**MMIO\_INSTALLPROC**

Adds an I/O procedure to the end of the MMPMMMIO.INI file. If an existing entry in the table matches the new entry, the new entry replaces the existing entry. An entry match is determined by specifying 0 or more of the match flags. If none are specified, the default is to match on the FOURCC code.

#### MMIO\_REMOVEPROC

Deletes a matching entry from the MMPMMMIO.INI file. An entry match is determined by specifying 0 or more of the match flags. If none are specified, the default is to match on the FOURCC code.

#### MMIO\_FINDPROC

Finds a matching entry from the MMPMMMIO.INI file. This fills in the remainder of the [MMINIFILEINFO](#) structure passed in the *pmminifileinfo* parameter. An entry match is determined by specifying 0 or more of the match flags. If none are specified, the default is to match on the FOURCC.

**Note:** If MMIO\_FINDFIRST is set, then MMIO\_FINDPROC does not default to the FOURCC.

#### MMIO\_MATCHFIRST

Finds the first entry in the MMPMMMIO.INI file if no match flags are specified. Otherwise, it finds the first entry that matches the contents of the fields specified by the match flags. In either case, the [MMINIFILEINFO](#) structure is returned.

#### MMIO\_MATCHNEXT

If no match flags are specified, this finds the next entry in the MMPMMMIO.INI file following the entry ([MMINIFILEINFO](#) structure) passed in the *pmminifileinfo* parameter. If match flags are specified, this finds the next entry that matches the search criteria specified by the flags. In either case, the [MMINIFILEINFO](#) structure is returned.

#### MMIO\_MATCHFOURCC

Use the FOURCC code (*fcc/OProc* field of the [MMINIFILEINFO](#) structure) as a search criteria.

#### MMIO\_MATCHDLL

Use the DLL Name (*szDLLName* field of the [MMINIFILEINFO](#) structure) as a search criteria.

#### MMIO\_MATCHPROCEDURENAME

Use the case-sensitive procedure name (*szProcName* field of the [MMINIFILEINFO](#) structure) as a search criteria.

#### MMIO\_FULLPATH

Use the drive or path given with the DLL name (*szDLLName* field of the [MMINIFILEINFO](#) structure), otherwise use only the base file name. This allows DLLs with the same base name to be loaded from different directories.

#### MMIO\_EXTENDED\_STRUCT

This flag *must* be set for release 1.1 and later releases. It indicates the expanded structure is being used.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

#### MMIO\_SUCCESS

If the function succeeds, 0 is returned.

#### MMIO\_ERROR

The function failed for a reason different from any of the following returns in this list.

#### MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

#### MMIOERR\_INTERNAL\_SYSTEM

An internal system error was found.

#### MMIOERR\_NO\_CORE

Unable to allocate enough memory for the MMPMMMIO.INI data.

#### MMIOERR\_INI\_OPEN

Unable to open the MMPMMMIO.INI file.

#### MMIOERR\_INVALID\_DLLNAME

Unable to validate the DLL name.

#### MMIOERR\_INVALID\_PROCEDURENAME

Unable to validate the procedure name.

#### MMIOERR\_MATCH\_NOT\_FOUND

Unable to find the I/O procedure to satisfy the input criteria.

-----

## mmioIniFileHandler - Remarks

The MMPMMMIO.INI file is in the directory specified in the MMBASE environment variable.

The DLL name (*szDLLName* field) specified in the [MMINIFILEINFO](#) structure must follow the same naming conventions as the `DosLoadModule` function. If the DLL or procedure name is invalid, an error is returned.

Any changes this function makes to the MMPMMMIO.INI file do not affect the MMIO internal data structures until the next time the DLL is loaded. This means that in the case of an addition, for example, the I/O procedure added is not active until the next time the MMIO.DLL is loaded.

In a deletion, the entry is removed and the entire file is rewritten to prevent it from growing with deleted entries. This is due to the way the OS/2 functions handle deleting entries from INI files in general. They do not reuse deleted entries.

If an error occurs during the loading of MMIO.DLL because the I/O procedure or procedure name cannot be validated, MMIO.DLL will still be loaded, but that particular I/O procedure will not be linked. The user must program for such situations.

**Note:** When MMIO services builds its internal structures for the I/O procedures, it processes the MMPMMMIO.INI file as a stack. The result is, the last I/O procedure in the file is the first one called during processing. Thus, in the MMPMMMIO.INI file, enter last those I/O procedures that you want to process first.

---

## mmioIniFileHandler - Related Functions

- [mmioInstallIOProc](#)

---

## mmioIniFileHandler - Example Code

The following code illustrates how to modify the initialization file for MMIO services.

```
MMINIFILEINFO mmIniFileInfo;
ULONG ulFlags = 0L;
ULONG rc;
...

memset( &mmIniFileInfo, '\0' sizeof(MMINIFILEINFO) );
mmIniFileInfo.fccIOProc = FOURCC_MYPROC;
strncpy( mmIniFileInfo.szDLLName, "MYPROC", DLLNAME_SIZE );
strncpy( mmIniFileInfo.szProcName, "MyIOProc", PROCNAME_SIZE );
ulFlags |= MMIO_INSTALLPROC;

rc = mmioIniFileHandler( &mmIniFileInfo,
                        ulFlags);

if (rc)
    /* error */
else
    ...
```

---

## mmioIniFileHandler - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

---

## mmioInstallIOProc

---

### mmioInstallIOProc - Syntax

This function installs an I/O procedure in the MMIO I/O procedure table, removes a procedure from the table, or finds a procedure when given its FOURCC identifier.

```
#define INCL_MMIOOS2
#include <os2.h>

FOURCC      fccIOProc; /* Four-character code. */
PMMIOPROC   pIOProc; /* IOProc address. */
ULONG       ulFlags; /* Flags. */
PMMIOPROC    rc; /* Address of the IOProc. */

rc = mmioInstallIOProc(fccIOProc, pIOProc,
    ulFlags);
```

---

### mmioInstallIOProc Parameter - fccIOProc

**fccIOProc** (**FOURCC**) - input

The four-character code of the I/O procedure to install, remove, or search for.

---

### mmioInstallIOProc Parameter - pIOProc

**pIOProc** (**PMMIOPROC**) - input

If this function is being called to install an I/O procedure, then *pIOProc* must contain the address of the I/O procedure entry point. Otherwise, *pIOProc* must be NULL.

---

### mmioInstallIOProc Parameter - ulFlags

**ulFlags** (**ULONG**) - input

Only one of the following flags can be set:

**MMIO\_INSTALLPROC**

Installs an I/O procedure with the entry point specified in the *pIOProc* parameter and the FOURCC in the *fccIOProc* parameter.

**MMIO\_REMOVEPROC**

Removes an I/O procedure (that has the FOURCC specified in the *fccIOProc* parameter) from the table of installed I/O procedures.

**MMIO\_FINDPROC**

Finds a previously installed I/O procedure with the identifier with the FOURCC specified in the *fccIOProc* parameter.

---

## mmioInstallIOProc Return Value - rc

**rc** (**PMMIOPROC**) - returns

Upon successful completion, this function returns the address of the I/O procedure that was installed, removed, or searched for. If a failure occurs, NULL is returned.

---

## mmioInstallIOProc - Parameters

**fccIOProc** (**FOURCC**) - input

The four-character code of the I/O procedure to install, remove, or search for.

**pIOProc** (**PMMIOPROC**) - input

If this function is being called to install an I/O procedure, then *pIOProc* must contain the address of the I/O procedure entry point. Otherwise, *pIOProc* must be NULL.

**ulFlags** (**ULONG**) - input

Only one of the following flags can be set:

**MMIO\_INSTALLPROC**

Installs an I/O procedure with the entry point specified in the *pIOProc* parameter and the FOURCC in the *fccIOProc* parameter.

**MMIO\_REMOVEPROC**

Removes an I/O procedure (that has the FOURCC specified in the *fccIOProc* parameter) from the table of installed I/O procedures.

**MMIO\_FINDPROC**

Finds a previously installed I/O procedure with the identifier with the FOURCC specified in the *fccIOProc* parameter.

**rc** (**PMMIOPROC**) - returns

Upon successful completion, this function returns the address of the I/O procedure that was installed, removed, or searched for. If a failure occurs, NULL is returned.

---

## mmioInstallIOProc - Remarks



Installing an I/O procedure in the MMIO I/O procedure table allows [mmioOpen](#) to call that procedure if the file name given to [mmioOpen](#) is specified as being a FOURCC of the same type specified in the *fcc/OProc* parameter. For example, if you install a hypothetical I/O procedure with the *fcc/OProc* parameter equal to XYZ, and then call [mmioOpen](#) to open the file FOO.XYZ, setting the *fcc/OProc* field of [MMIOINFO](#) = XYZ, your I/O procedure is called to open and perform I/O on FOO.XYZ.

[mmioInstallIOProc](#) maintains a separate list of installed I/O procedures for each OS/2 application that uses MMIO services. If application X (or a DLL that application X calls) installs an I/O procedure identified as ABC, and application Y (or a DLL that Y calls) installs another I/O procedure identified as ABC, then MMIO services keeps separate entries in the I/O procedure table. Therefore, different applications can use the same I/O procedure identifier for different I/O procedures without conflict. Also, if an I/O procedure is implemented in a DLL and shared among several applications, each application must call [mmioInstallIOProc](#) individually (or get the DLL to call it for the application), once to install the I/O procedure, and once to remove it from the table.

If an application calls [mmioInstallIOProc](#) more than once to register the same I/O procedure, then [mmioInstallIOProc](#) must be called once with [MMIO\\_REMOVEPROC](#) for each time it is called with [MMIO\\_INSTALLPROC](#).

[mmioInstallIOProc](#) will not prevent an application from installing two different I/O procedures with the same identifier, or installing an I/O procedure with the same identifier as a built-in I/O procedure (DOS, MEM, or BND). The most recently installed procedure takes precedence, and is the first one to get removed by [MMIO\\_REMOVEPROC](#).

---

## mmioInstallIOProc - Related Functions

- [mmioSendMessage](#)
- 

## mmioInstallIOProc - Example Code

The following code illustrates how to install an IOProc.

```
FOURCC    fourcc1 ;
PMMIOPROC pIOProc1;
ULONG     ulFlags;
.....

    fourcc1= FOURCC_WAVE;
    ulFlags= MMIO_FINDPROC;
    pIOProc1= mmioInstallIOProc (fourcc1, pIOProc1, ulFlags);
    if (!pIOProc1)
        /* WAVE I/O Procedure NOT FOUND */
    else
        /* WAVE I/O Procedure FOUND */
    .....
```

The following is a procedure prototype for a standard I/O procedure call.

```
LONG APIENTRY MMIOPROC (PVOID pmmioinfo, USHORT usMsg,
                        LONG lParam1, LONG lParam2);
```

The following code illustrates how to call a custom I/O procedure and what parameters are required.

```
(LONG) lrc= IOProc(pmmioinfo, usMsg, lParam1, lParam2);
```

The I/O procedure must return [MMIOERR\\_UNSUPPORTED\\_MESSAGE](#) if it or subsequent I/O procedures it has called through [mmioSendMessage](#) does not understand *usMsg*.

---

# mmioInstallIOProc - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

---

## mmioLoadCODECProc

---

## mmioLoadCODECProc - Syntax

This function loads the CODEC Proc installed in the MMPMMMIO.INI file and returns the entry point. It is normally issued by the IOProc to access the CODEC Proc.

```
#define INCL_MMIOOS2
#define INCL_MMIO_CODEC
#include <os2.h>

PCODECINIFILEINFO  pCODECIniFileInfo; /* Pointer to CODECINIFILEINFO. */
PHMODULE           phMod;             /* Pointer. */
ULONG              ulFlags;           /* Flags. */
PCODECPROC         rc;                /* Address of CODEC Proc. */

rc = mmioLoadCODECProc(pCODECIniFileInfo,
                      phMod, ulFlags);
```

---

## mmioLoadCODECProc Parameter - pCODECIniFileInfo

**pCODECIniFileInfo** ([PCODECINIFILEINFO](#)) - input

Pointer to a structure containing the CODEC information. The search parameters used to load the CODEC procedure are specified in the *ulFlags* parameter.

---

## mmioLoadCODECProc Parameter - phMod

**phMod** ([PHMODULE](#)) - output

Pointer to the returned module handle of the loaded CODEC procedure.

---

## mmioLoadCODECProc Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input

Specifies options for the operation. Contains one of the following flags:

**MMIO\_MATCHCOMPRESSTYPE**

Uses compression type (*ulCompressType* field of [CODECINIFILEINFO](#)) as a search criteria for loading the CODEC procedure.

**MMIO\_MATCHCOMPRESSSUBTYPE**

Uses the compression subtype (*ulCompressSubType* field of [CODECINIFILEINFO](#)) as a search criteria for loading the CODEC procedure.

**MMIO\_MATCHHWID**

Uses the hardware ID (*szHWID* field of [CODECINIFILEINFO](#)) as a search criteria for loading the CODEC procedure.

**MMIO\_MATCHCAPSFLAGS**

Uses the capability flags (*ulCapsFlags* field of [CODECINIFILEINFO](#)) as a search criteria for loading the CODEC procedure. This is not based on an exact match. If the target entry contains the flags, the match is satisfied.

**MMIO\_SKIPMATCH**

Skips the search and loads the CODEC procedure using the DLL name (*szDLLName*) and Procedure name (*szProcName*) specified in the [CODECINIFILEINFO](#) structure.

**MMIO\_MATCHDLL**

Uses the DLL Name (*szDLLName* field of [CODECINIFILEINFO](#)) as a search criteria for loading the CODEC procedure.

**MMIO\_MATCHFOURCC**

Uses the FOURCC code (*fcc* field of [CODECINIFILEINFO](#)) as a search criteria for loading the CODEC procedure.

**MMIO\_MATCHPROCEDURENAME**

Uses the case-sensitive procedure name (*szProcName* field of [CODECINIFILEINFO](#)) as a search criteria for loading the CODEC procedure.

---

## mmioLoadCODECProc Return Value - rc

**rc** ([PCODECPROC](#)) - returns

Upon successful completion, this function returns the address of the CODEC procedure that has been loaded. If a failure occurs, NULL is returned.

---

## mmioLoadCODECProc - Parameters

**pCODECIniFileInfo** ([PCODECINIFILEINFO](#)) - input

Pointer to a structure containing the CODEC information. The search parameters used to load the CODEC procedure are specified in the *ulFlags* parameter.

**phMod** (**PHMODULE**) - output

Pointer to the returned module handle of the loaded CODEC procedure.

**ulFlags** (**ULONG**) - input

Specifies options for the operation. Contains one of the following flags:

**MMIO\_MATCHCOMPRESSTYPE**

Uses compression type (*ulCompressType* field of **CODECINIFILEINFO**) as a search criteria for loading the CODEC procedure.

**MMIO\_MATCHCOMPRESSSUBTYPE**

Uses the compression subtype (*ulCompressSubType* field of **CODECINIFILEINFO**) as a search criteria for loading the CODEC procedure.

**MMIO\_MATCHHHWID**

Uses the hardware ID (*szHWID* field of **CODECINIFILEINFO**) as a search criteria for loading the CODEC procedure.

**MMIO\_MATCHCAPSFLAGS**

Uses the capability flags (*ulCapsFlags* field of **CODECINIFILEINFO**) as a search criteria for loading the CODEC procedure. This is not based on an exact match. If the target entry contains the flags, the match is satisfied.

**MMIO\_SKIPMATCH**

Skips the search and loads the CODEC procedure using the DLL name (*szDLLName*) and Procedure name (*szProcName*) specified in the **CODECINIFILEINFO** structure.

**MMIO\_MATCHDLL**

Uses the DLL Name (*szDLLName* field of **CODECINIFILEINFO**) as a search criteria for loading the CODEC procedure.

**MMIO\_MATCHFOURCC**

Uses the FOURCC code (*fcc* field of **CODECINIFILEINFO**) as a search criteria for loading the CODEC procedure.

**MMIO\_MATCHPROCEDURENAME**

Uses the case-sensitive procedure name (*szProcName* field of **CODECINIFILEINFO**) as a search criteria for loading the CODEC procedure.

**rc** (**PCODECPROC**) - returns

Upon successful completion, this function returns the address of the CODEC procedure that has been loaded. If a failure occurs, NULL is returned.

-----

## mmioLoadCODECProc - Remarks

If none of the flags are specified, the default is to match on the FOURCC.

The following is a procedure prototype for a standard CODEC procedure call.

```
PCODECPROC APIENTRY mmioLoadCODECProc
    (PCODECINIFILEINFO pCODECIniFileInfo,
     PHMODULE phMod, ULONG ulFlags);
```

The following example shows how to call a CODEC Proc and the required parameters.

```
LONG APIENTRY CODECProc (phCODEC, usMsg, lParam1, lParam2)
```

Note that **CODECProc** represents the entry point of a CODEC DLL.

**phCODEC** (**PHCODEC**) - Input/Output:

Pointer to a CODEC instance handle returned by MMIOM\_CODEC\_OPEN.

**usMsg(USHORT)** - Input

The message that the CODEC Proc is asked to process. The following predefined messages are documented separately.

```
MMIOM_CODEC_OPEN
MMIOM_CODEC_CLOSE
MMIOM_CODEC_COMPRESS
MMIOM_CODEC_DECOMPRESS
MMIOM_CODEC_QUERYNAME
MMIOM_CODEC_QUERYNAMELENGTH
```

**lParam1(LONG)** - input/output

Specifies additional message information.

**lParam2(LONG)** - input/output

Specifies additional message information.

---

## mmioLoadCODECProc - Example Code

The following code illustrates how to load a CODEC Proc.

```
CODECINIFILEINFO codecIniFileInfo;
HMODULE hMod;
PCODECPROC pCODECProc;
ULONG rc, ulFlags;
...
memset(&codecIniFileInfo, '\0', sizeof(CODECINIFILEINFO));
codecIniFileInfo.ulStructlen = sizeof(CODECINIFILEINFO);
codecIniFileInfo.fcc = FOURCC_MYPROC;
codecIniFileInfo.ulCompressType = COMPRESSTYPE_MYPROC;
codecIniFileInfo.ulCompressSubType = COMPRESSSUBTYPE_MYPROC;
codecIniFileInfo.ulCapsFlags = CODEC_CAN_DECOMPRESS;
codecIniFileInfo.szHWID = HWID_MYPROC;
ulFlags = MMIO_MATCHFOURCC|
          MMIO_MATCHCOMPRESSTYPE|
          MMIO_MATCHCOMPRESSSUBTYPE|
          MMIO_MATCHHWID|
          MMIO_MATCHCAPSFLAGS;
pCODECProc = mmioLoadCODECProc (&codecIniFileInfo, &hMod, ulFlags);

if (!pCODECProc)
    /* error */
else
    ...
```

---

## mmioLoadCODECProc - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

---

## mmioOpen

---

### mmioOpen - Syntax

This function opens a file for unbuffered I/O or buffered I/O (including RIFF I/O). The file may be a DOS file, a memory file, or an element of a custom storage system (provided that a custom I/O procedure has been installed using [mmioInstallIOProc](#)).

```
#define INCL_MMIOOS2
#include <os2.h>

PSZ      pszFileName; /* File name. */
PMMIOINFO pmmioinfo; /* Pointer to MMIOINFO. */
ULONG     ulOpenFlags; /* Flags. */
HMMIO     hmmio;       /* MMIO file handle. */

hmmio = mmioOpen(pszFileName, pmmioinfo, ulOpenFlags);
```

---

### mmioOpen Parameter - pszFileName

#### **pszFileName (PSZ)** - input

The name of the file to open. If the *fccIOProc* field of [MMIOINFO](#) is NULL, mmioOpen looks at the *pszFileName* parameter to figure out what kind of file to open, as follows:

- If the *pszFileName* parameter does not contain a plus (+), the name is assumed to be that of a DOS file, which is opened using the file system file open process.
  - If the file name is of the form ABC.EXT+ELEMENTNAME, the extension EXT is assumed to identify an installed I/O procedure that is called to perform I/O on the file (see [mmioInstallIOProc](#)). If the extension is BND, the system-provided BND I/O procedure processes the open. Note also that ELEMENTNAME could be of the form ABC.EXT followed by a plus sign (+). Parsing of the file name is done from right to left, so the first I/O procedure called belongs to the rightmost extension name that is followed by the +. The I/O procedure must be already installed and be able to further parse the file name, if required. The trailing separator character is stripped off by mmioOpen, and is not passed to the I/O procedure.
  - If the *pszFileName* parameter is NULL, then the *auIInfo* field of [MMIOINFO](#) contains the DOS file handle, and I/O is performed on that file handle. The MMIO offset is the same as the DOS offset when mmioOpen is called.
  - The *pszFileName* parameter cannot be longer than 260 bytes, including the terminating NULL, for a fully-qualified path name, or 256 bytes for an individual component name, including the terminating NULL.
- 

### mmioOpen Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#)) - input

A pointer to a caller-provided [MMIOINFO](#) structure containing extra parameters used by mmioOpen. Only certain values of the [MMIOINFO](#) structure may be used as input to the mmioOpen function. These fields are *fccIOProc*, *pIOProc*, *cchBuffer*, *pchBuffer*, *aullInfo*, and *ulTranslate*. *fccIOProc* and *pIOProc* are used if you wish to identify the I/O procedure to be used, rather than allowing MMIO to determine the appropriate I/O procedure. *cchBuffer* and *pchBuffer* are used for buffered access. *aullInfo* can contain a media type which restricts the open to I/O procedures of that type. *ulTranslate* is used to specify whether or not translation of header and data is performed.

The *pmmioinfo* parameter can be NULL if the default values of the fields of [MMIOINFO](#) are sufficient. All unused fields must be set to 0, including reserved fields. (The easiest way to do this is to fill the structure with NULL bytes before setting the desired fields.) See the [MMIOINFO](#) data structure for more information. mmioOpen will modify the *ulErrorRet* field of [MMIOINFO](#) if an error is encountered.

---

## mmioOpen Parameter - ulOpenFlags

**ulOpenFlags** ([ULONG](#)) - input

Contains one or more of the following flags:

**Note:** The MMIO\_READ, MMIO\_WRITE, and MMIO\_READWRITE flags are mutually exclusive.

MMIO\_READ

Opens the file for reading only. This is the default behavior if MMIO\_WRITE and MMIO\_READWRITE are not specified. However, the flag is not automatically set in the default case.

MMIO\_WRITE

Opens the file for writing. You should not read from a file opened in this way.

MMIO\_READWRITE

Opens the file for both reading and writing.

**Note:** To save a wave file, you must open the file with the MMIO\_READWRITE flag. After the data is written, the I/O procedure will need to descend back into the wave chunk-a process that requires read support.

MMIO\_BUFSHARED

Requests that if MMIO services allocates the I/O buffer, it does so from shared memory.

MMIO\_VERTBAR

Requests that the vertical bar symbol (|) rather than the plus sign (+) be used as a file separator.

MMIO\_EXCLUSIVE

Opens the file with exclusive mode, denying other processes both read and write access to the file. mmioOpen fails if the file has been opened in any other mode for read or write access, even by the current process.

MMIO\_DENYWRITE

Opens the file and denies other processes write access. mmioOpen fails if the file has been opened in compatibility or for write access by any other process.

MMIO\_DENYREAD

Opens the file and denies other processes read access. mmioOpen fails if the file has been opened in compatibility or for read access by any other process.

MMIO\_DENYNONE

Opens the file and denies other processes read access. mmioOpen fails if the file has been opened in compatibility or for read access by any other process. This is the default if no share mode flags are defined.

MMIO\_CREATE

Directs mmioOpen to create a new file. If the file already exists, it is truncated to 0 length. For a memory file, MMIO\_CREATE indicates the end of the file is initially at the start of the buffer.

MMIO\_DELETE

Directs mmioOpen to delete the file. The *pszFileName* parameter should not be NULL. The return value will be TRUE (sent to *hmmio*) if the file was deleted successfully, FALSE otherwise. Do not call [mmioClose](#) if MMIO\_DELETE has been specified. All other flags are ignored if MMIO\_DELETE is specified.

#### MMIO\_ALLOCBUF

Directs mmioOpen to allocate an I/O buffer. If the *cchBuffer* field of **MMIOINFO** is 0, then a default buffer size (specified by the constant MMIO\_DEFAULTBUFFER) is used. If the caller provides an I/O buffer, then MMIO\_ALLOCBUF should not be specified.

#### MMIO\_APPEND

Directs mmioOpen to allow appending to the end of the file. This will cause the logical file pointer to be positioned at the end of file when the open process completes. The open fails if both MMIO\_CREATE and MMIO\_APPEND are set. In the case of a BND element, this flag allows the element to expand past its existing fixed boundary by deleting the existing element and rewriting it at the end of a compound-file resource group (CGRP).

#### MMIO\_NOIDENTIFY

Directs mmioOpen to directly open the file without attempting to automatically identify the file. An automatic identify is the default for this function.

-----

## mmioOpen Return Value - hmmio

#### hmmio (**HMMIO**) - returns

A handle is returned to use with further calls to MMIO functions to perform I/O. This handle is not a file system handle. Do not use this with such operations as OS/2 file system read, or write.

NULL is returned if the file cannot be opened. See the exception for the preceding *uiOpenFlags* parameter flag MMIO\_DELETE. If the *pmmioinfo* parameter is not NULL, the *uiErrorRet* field of its **MMIOINFO** structure will contain extended error information returned by the I/O procedure. If delete fails, *uiErrorRet* contains MMIOERR\_DELETE\_FAILED. The error return can also be queried by calling the [mmioGetLastError](#) function. See [Return Codes](#) for a description of MMIO Manager error codes.

-----

## mmioOpen - Parameters

#### pszFileName (**PSZ**) - input

The name of the file to open. If the *fccIOProc* field of **MMIOINFO** is NULL, mmioOpen looks at the *pszFileName* parameter to figure out what kind of file to open, as follows:

- If the *pszFileName* parameter does not contain a plus (+), the name is assumed to be that of a DOS file, which is opened using the file system file open process.
- If the file name is of the form ABC.EXT+ELEMENTNAME, the extension EXT is assumed to identify an installed I/O procedure that is called to perform I/O on the file (see [mmioInstallIOProc](#)). If the extension is BND, the system-provided BND I/O procedure processes the open. Note also that ELEMENTNAME could be of the form ABC.EXT followed by a plus sign (+). Parsing of the file name is done from right to left, so the first I/O procedure called belongs to the rightmost extension name that is followed by the +. The I/O procedure must be already installed and be able to further parse the file name, if required. The trailing separator character is stripped off by mmioOpen, and is not passed to the I/O procedure.
- If the *pszFileName* parameter is NULL, then the *auIInfo* field of **MMIOINFO** contains the DOS file handle, and I/O is performed on that file handle. The MMIO offset is the same as the DOS offset when mmioOpen is called.
- The *pszFileName* parameter cannot be longer than 260 bytes, including the terminating NULL, for a fully-qualified path name, or 256 bytes for an individual component name, including the terminating NULL.

#### pmmioinfo (**PMMIOINFO**) - input

A pointer to a caller-provided **MMIOINFO** structure containing extra parameters used by mmioOpen. Only certain values of the **MMIOINFO** structure may be used as input to the mmioOpen function. These fields are *fccIOProc*, *pIOProc*, *cchBuffer*, *pchBuffer*, *auIInfo*, and *uiTranslate*. *fccIOProc* and *pIOProc* are used if you wish to identify the I/O procedure to be used, rather than allowing MMIO to determine the appropriate I/O procedure. *cchBuffer* and *pchBuffer* are used for buffered access. *auIInfo* can contain a media type which restricts the open to I/O procedures of that type. *uiTranslate* is used to specify whether or not translation of header and data is performed.

The *pmmioinfo* parameter can be NULL if the default values of the fields of **MMIOINFO** are sufficient. All unused fields must be set to



0, including reserved fields. (The easiest way to do this is to fill the structure with NULL bytes before setting the desired fields.) See the [MMIOINFO](#) data structure for more information. `mmioOpen` will modify the *ulErrorRet* field of [MMIOINFO](#) if an error is encountered.

**ulOpenFlags** ([ULONG](#)) - input

Contains one or more of the following flags:

**Note:** The `MMIO_READ`, `MMIO_WRITE`, and `MMIO_READWRITE` flags are mutually exclusive.

`MMIO_READ`

Opens the file for reading only. This is the default behavior if `MMIO_WRITE` and `MMIO_READWRITE` are not specified. However, the flag is not automatically set in the default case.

`MMIO_WRITE`

Opens the file for writing. You should not read from a file opened in this way.

`MMIO_READWRITE`

Opens the file for both reading and writing.

**Note:** To save a wave file, you must open the file with the `MMIO_READWRITE` flag. After the data is written, the I/O procedure will need to descend back into the wave chunk-a process that requires read support.

`MMIO_BUFSHARED`

Requests that if MMIO services allocates the I/O buffer, it does so from shared memory.

`MMIO_VERTBAR`

Requests that the vertical bar symbol (|) rather than the plus sign (+) be used as a file separator.

`MMIO_EXCLUSIVE`

Opens the file with exclusive mode, denying other processes both read and write access to the file. `mmioOpen` fails if the file has been opened in any other mode for read or write access, even by the current process.

`MMIO_DENYWRITE`

Opens the file and denies other processes write access. `mmioOpen` fails if the file has been opened in compatibility or for write access by any other process.

`MMIO_DENYREAD`

Opens the file and denies other processes read access. `mmioOpen` fails if the file has been opened in compatibility or for read access by any other process.

`MMIO_DENYNONE`

Opens the file and denies other processes read access. `mmioOpen` fails if the file has been opened in compatibility or for read access by any other process. This is the default if no share mode flags are defined.

`MMIO_CREATE`

Directs `mmioOpen` to create a new file. If the file already exists, it is truncated to 0 length. For a memory file, `MMIO_CREATE` indicates the end of the file is initially at the start of the buffer.

`MMIO_DELETE`

Directs `mmioOpen` to delete the file. The *pszFileName* parameter should not be NULL. The return value will be TRUE (sent to *hmmio*) if the file was deleted successfully, FALSE otherwise. Do not call [mmioClose](#) if `MMIO_DELETE` has been specified. All other flags are ignored if `MMIO_DELETE` is specified.

`MMIO_ALLOCBUF`

Directs `mmioOpen` to allocate an I/O buffer. If the *cchBuffer* field of [MMIOINFO](#) is 0, then a default buffer size (specified by the constant `MMIO_DEFAULTBUFFER`) is used. If the caller provides an I/O buffer, then `MMIO_ALLOCBUF` should not be specified.

`MMIO_APPEND`

Directs `mmioOpen` to allow appending to the end of the file. This will cause the logical file pointer to be positioned at the end of file when the open process completes. The open fails if both `MMIO_CREATE` and `MMIO_APPEND` are set. In the case of a BND element, this flag allows the element to expand past its existing fixed boundary by deleting the existing element and rewriting it at the end of a compound-file resource group (CGRP).

`MMIO_NOIDENTIFY`

Directs `mmioOpen` to directly open the file without attempting to automatically identify the file. An automatic identify is the default for this function.

**hmmio** ([HMMIO](#)) - returns

A handle is returned to use with further calls to MMIO functions to perform I/O. This handle is not a file system handle. Do not use this with such operations as OS/2 file system read, or write.

NULL is returned if the file cannot be opened. See the exception for the preceding *ulOpenFlags* parameter flag MMIO\_DELETE. If the *pmmioinfo* parameter is not NULL, the *ulErrorRet* field of its MMIOINFO structure will contain extended error information returned by the I/O procedure. If delete fails, *ulErrorRet* contains MMIOERR\_DELETE\_FAILED. The error return can also be queried by calling the [mmioGetLastError](#) function. See [Return Codes](#) for a description of MMIO Manager error codes.

---

## mmioOpen - Remarks

If the *pmmioinfo* parameter is provided the following fields must be filled in by the caller as described:

*fccIOProc* - If this field is not NULL, it is the four character code of an installed I/O procedure that will handle I/O. If *fccIOProc* and *pIOProc* are NULL, mmioOpen determines which I/O procedure to use based on the syntax of the *pszFileName* parameter. (See description of *pszFileName*.) If *fccIOProc* is NULL, but *pIOProc* is not NULL, the custom I/O procedure (*pIOProc*) is used. This I/O procedure does not need to be installed using [mmioInstallIOProc](#).

The following I/O procedure identifiers are defined:

FOURCC_DOS	<i>pszFileName</i> is assumed to be either the name of a DOS file (which is to be opened using the file system opening procedure), or <i>auInfo</i> contains the DOS file handle of an open file handle (directed to a <a href="#">PSZ</a> ).
FOURCC_BND	<p>A RIFF compound file element is opened. This procedure calls mmioCFOpen if necessary to read the CTOC into memory before the element can be accessed.</p> <p>If MMIO_CREATE or MMIO_APPEND is specified when opening an element, the system automatically accesses the element as exclusive until the element is closed.</p>
FOURCC_MEM	<p>A memory file is opened. The <i>pszFileName</i> parameter should be NULL. There are two ways to set up a memory file:</p> <ol style="list-style-type: none"><li>1. The <i>pchBuffer</i> field points to a caller-supplied memory buffer, and the <i>cchBuffer</i> field indicates the size of the buffer. The memory file can be read and written like an ordinary file, but the file can not be expanded larger than the number of bytes specified in <i>cchBuffer</i>. If the MMIO_CREATE flag is specified, the end of the file is initially at the beginning of the buffer. If MMIO_CREATE is NULL, the user specifies in <i>auInfo[1]</i> the number of bytes of data in the memory buffer. For the default case, where <i>auInfo[1]</i> is 0, the end of the file is set to the end of the buffer.</li><li>2. mmioOpen can allocate the memory block for the memory file. The <i>cchBuffer</i> field is the desired initial size of the memory I/O buffer. The <i>auInfo[0]</i> field must be the number of bytes by which to expand the memory file if the initial buffer becomes filled. The MMIO_CREATE flag must be specified. The end of the file is initially at the beginning of the buffer, and if the memory file must be expanded, it is expanded at least <i>auInfo[0]</i> bytes at a time. If <i>auInfo[0]</i> is 0, the buffer cannot expand. There is no default for <i>cchBuffer</i> when used to open a memory file.</li></ol>

The *pIOProc* field uses a custom I/O procedure defined in this field. Set the *fccIOProc* field to NULL, and set the *pIOProc* field to the address of the custom I/O procedure to use. Otherwise, *pIOProc* must be zero.

*cchBuffer* specifies the size of the memory block to use as an I/O buffer or as a memory file. See descriptions of *pchBuffer* and the MMIO\_ALLOCBUF flag for more information.

The *pchBuffer* field points to a caller-provided memory buffer to use as an I/O buffer or as a memory file. The *cchBuffer* field must be the size of the buffer. If the caller-provided memory buffer is not provided, *pchBuffer* must be NULL.

To open a memory file that performs I/O on an already allocated memory block, set the *pszFileName* parameter to NULL, the *fccIOProc* field to FOURCC\_MEM, the *pchBuffer* field to point to the memory buffer, the *cchBuffer* field to the size of the memory buffer, the *ulOpenFlags* parameter to MMIO\_READWRITE (plus MMIO\_CREATE if the memory file is initially empty), and set all other fields of the MMIOINFO structure passed in the *pmmioinfo* parameter to zero.

For example, to open a memory file that is initially 32KB in size, but can be expanded at least 16KB at a time:

1. Set *auInfo[0]* = 16K
2. Set *cBytes* = 16K
3. Set *pszFileName* to NULL
4. Set *fccIOProc* to FOURCC\_MEM

5. Set *ulOpenFlags* to indicate MMIO\_READWRITE plus MMIO\_CREATE
6. Set other fields of *pmmioinfo* to zero

Initially this file will be empty.

A system-allocated memory buffer must be opened as MMIO\_READWRITE, which is the default for that case. If this does not happen, the open-a-memory file process fails.

If both a user buffer is specified, and an expansion size is requested, the open-a-memory file process fails because it is not possible to later expand the buffer size in this situation.

As with DOS file handles, different applications cannot share a single *hmmio*. In other words, MMIO handles ([HMMIO](#)) are unique to a process.

---

## mmioOpen - Related Functions

- [mmioClose](#)
- [mmioRead](#)
- [mmioSeek](#)
- [mmioWrite](#)

---

## mmioOpen - Example Code

The following code illustrates how to open a file for I/O.

```
HMMIO hmmiol;
MMIOINFO mmioinfo;
ULONG ulFlags = 0L;
...

memset( &mmioinfo, '\0', sizeof(MMIOINFO) );
mmioinfo.fccIOProc = FOURCC_WAVE;
ulFlags |= MMIO_READ | MMIO_DENYNONE;

hmmiol = mmioOpen("sounds.bnd+train.wav", &mmioinfo, ulFlags);
if (!hmmiol)
    /* error */
else
    ...
```

---

## mmioOpen - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

# mmioQueryCODECName

---

## mmioQueryCODECName - Syntax

This function returns the CODEC procedure name.

```
#define INCL_MMIOOS2
#define INCL_MMIO_CODEC
#include <os2.h>

PCODECINIFILEINFO  pCODECIniFileinfo; /* Pointer. */
PSZ                pszCODECName;      /* Pointer. */
PULONG             pulBytesRead;      /* Byte size read. */
ULONG              rc;                /* Return codes. */

rc = mmioQueryCODECName(pCODECIniFileinfo,
                        pszCODECName, pulBytesRead);
```

---

## mmioQueryCODECName Parameter - pCODECIniFileinfo

**pCODECIniFileinfo** ([PCODECINIFILEINFO](#)) - input

Pointer to the [PCODECINIFILEINFO](#) data structure containing the CODEC information. Only the *fcc*, *ulCompressType*, *ulCompressSubType*, *szHWID*, and *ulCapsFlags* fields of the structure are used to identify a CODEC procedure.

---

## mmioQueryCODECName Parameter - pszCODECName

**pszCODECName** ([PSZ](#)) - output

Pointer to the CODEC name. The function fills in the CODEC name associated with the specified CODEC procedure, up to *pulBytesRead* bytes. Make sure the buffer is at least one byte long.

---

## mmioQueryCODECName Parameter - pulBytesRead

**pulBytesRead** ([PULONG](#)) - in/out

On input, specifies the size in bytes of the *pszCODECName*. On output, returns the number of bytes read into the *pszCODECName*.

---

# mmioQueryCODECName Return Value - rc

**rc** ([ULONG](#)) - returns

Return codes.

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The function failed for a reason different from any of the returns in this list.

MMIO\_INVALID\_PARAMETER

An invalid parameter was passed.

-----

## mmioQueryCODECName - Parameters

**pCODECIniFileInfo** ([PCODECINIFILEINFO](#)) - input

Pointer to the [CODECINIFILEINFO](#) data structure containing the CODEC information. Only the *fcc*, *ulCompressType*, *ulCompressSubType*, *szHWID*, and *ulCapsFlags* fields of the structure are used to identify a CODEC procedure.

**pszCODECName** ([PSZ](#)) - output

Pointer to the CODEC name. The function fills in the CODEC name associated with the specified CODEC procedure, up to *pulBytesRead* bytes. Make sure the buffer is at least one byte long.

**pulBytesRead** ([PULONG](#)) - in/out

On input, specifies the size in bytes of the *pszCODECName*. On output, returns the number of bytes read into the *pszCODECName*.

**rc** ([ULONG](#)) - returns

Return codes.

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The function failed for a reason different from any of the returns in this list.

MMIO\_INVALID\_PARAMETER

An invalid parameter was passed.

-----

## mmioQueryCODECName - Example Code

The following code illustrates how to obtain the name of the CODEC procedure.

```
CODECINIFILFO codecIniFileInfo;
PULONG pulBytesRead;
PSZ pszCODECName;
ULONG rc;
...

memset(&codecIniFileInfo, '\0', sizeof(CODECINIFILEINFO);
codecIniFileInfo.ulStructLen = sizeof(CODECINIFILEINFO);
codecIniFileInfo.fcc = FOURCC_MYPROC;
codecIniFileInfo.ulCompressType = COMPRESS_TYPE_MYPROC
codecIniFileInfo.ulCompressSubType = COMPRESSSUBTYPE_MYPROC
```

```

codecIniFileInfo.ulCapsFlags = CODEC_CAN_DECOMPRESS;
codecIniFileInfo.szHWID = MYPROC;
rc = mmioQueryCODECNameLength (&codecIniFileInfo,
    pulBytesRead);
if (rc)
    /* error */
else
    pszCODECName = malloc(*pulBytesRead + sizeof (char));
rc = mmioQueryCODECName ( &codecIniFileInfo,
    pszCODECName,
    pulBytesRead);

if (rc)
    /* error */
else
}
...

```

---

## mmioQueryCODECName - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Example Code](#)  
[Glossary](#)

---

## mmioQueryCODECNameLength

---

## mmioQueryCODECNameLength - Syntax

This function returns the length of the CODEC procedure name.

```

#define INCL_MMIOOS2
#define INCL_MMIO_CODEC
#include <os2.h>

PCODECINIFILEINFO    pCODECIniFileinfo; /* Pointer to CODECINIFILEINFO structure. */
PULONG               pulNameLength;      /* # of bytes. */
ULONG                rc;                  /* Return codes. */

rc = mmioQueryCODECNameLength(pCODECIniFileinfo,
    pulNameLength);

```

---

## mmioQueryCODECNameLength Parameter -

# pCODECIniFileInfo

**pCODECIniFileInfo** ([PCODECINIFILEINFO](#)) - input

Pointer to the [CODECINIFILEINFO](#) data structure containing the CODEC information. Only the *fcc*, *ulCompressType*, *ulCompressSubType*, *szHWID*, and *ulCapsFlags* fields of the structure are used to identify a CODEC procedure.

-----

## mmioQueryCODECNameLength Parameter - pulNameLength

**pulNameLength** ([PULONG](#)) - output

Number of bytes in the CODEC procedure name is returned. The returned length does not include the NULL terminating character.

-----

## mmioQueryCODECNameLength Return Value - rc

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The function failed for a reason different from any of the following returns in this list.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

-----

## mmioQueryCODECNameLength - Parameters

**pCODECIniFileInfo** ([PCODECINIFILEINFO](#)) - input

Pointer to the [CODECINIFILEINFO](#) data structure containing the CODEC information. Only the *fcc*, *ulCompressType*, *ulCompressSubType*, *szHWID*, and *ulCapsFlags* fields of the structure are used to identify a CODEC procedure.

**pulNameLength** ([PULONG](#)) - output

Number of bytes in the CODEC procedure name is returned. The returned length does not include the NULL terminating character.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The function failed for a reason different from any of the following returns in this list.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

-----

# mmioQueryCODECNameLength - Example Code

The following code illustrates how to obtain the length of the CODEC procedure name.

```
CODECINIFILFO CODECIniFileInfo;
PULONG pulNameLength;
...

memset( &CODECIniFileInfo, '\0', sizeof(CODECINIFILEINFO) );
CODECIniFileInfo.ulStructLen = sizeof(CODECINIFILEINFO);
CODECIniFileInfo.fcc = FOURCC_MYPROC;
CODECIniFileInfo.ulCompressType = COMPRESSTYPE_MYPROC;
CODECIniFileInfo.ulCompressSubType = COMPRESSSUBTYPE_MYPROC;
CODECIniFileInfo.ulMediaType = MEDIATYPE_MYPROC;
CODECIniFileInfo.ulCapsFlags = CODEC_CAN_DECOMPRESS;
CODECIniFileInfo.szHWID = HWID_MYPROC
rc = mmioQueryCODECNameLength (&CODECIniFileInfo,
                               pulNameLength);

if (rc)
    /* error */
else
    ...
```

---

## mmioQueryCODECNameLength - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Example Code](#)  
[Glossary](#)

---

## mmioQueryFormatCount

---

## mmioQueryFormatCount - Syntax

This function provides the number of I/O procedures that match the specified data format.

```
#define INCL_MMIOOS2
#include <os2.h>

PMMFORMATINFO pmmformatinfo; /* Indicates field type. */
PLONG plNumFormats; /* Number of supported formats. */
ULONG ulReserved; /* Reserved. */
ULONG ulFlags; /* Reserved. */
ULONG rc; /* Return codes. */

rc = mmioQueryFormatCount(pmmformatinfo, plNumFormats,
```



```
ulReserved, ulFlags);
```

-----

## mmioQueryFormatCount Parameter - pmmformatinfo

**pmmformatinfo** ([PMMFORMATINFO](#)) - input

Indicates a specific IOProc that is to be queried on the basis of whether the *fccIOProc* field or the *ulMediaType* field or both are specified in the structure. If neither is set, all I/O procedures are counted (queried).

-----

## mmioQueryFormatCount Parameter - plNumFormats

**plNumFormats** ([PLONG](#)) - output

Pointer to a LONG. Returns the number of formats supported.

-----

## mmioQueryFormatCount Parameter - ulReserved

**ulReserved** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

-----

## mmioQueryFormatCount Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

-----

## mmioQueryFormatCount Return Value - rc

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The function failed for a reason different from any of the following returns in this list.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_INTERNAL\_SYSTEM  
An internal system error occurred.

---

## mmioQueryFormatCount - Parameters

**pmmformatinfo** ([PMMFORMATINFO](#)) - input  
Indicates a specific IOProc that is to be queried on the basis of whether the *fccIOProc* field or the *ulMediaType* field or both are specified in the structure. If neither is set, all I/O procedures are counted (queried).

**plNumFormats** ([PLONG](#)) - output  
Pointer to a LONG. Returns the number of formats supported.

**ulReserved** ([ULONG](#)) - input  
Reserved for future use and must be set to zero.

**ulFlags** ([ULONG](#)) - input  
Reserved for future use and must be set to zero.

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure:

MMIO\_SUCCESS  
If the function succeeds, 0 is returned.

MMIO\_ERROR  
The function failed for a reason different from any of the following returns in this list.

MMIOERR\_INVALID\_PARAMETER  
An invalid parameter was passed.

MMIOERR\_INTERNAL\_SYSTEM  
An internal system error occurred.

---

## mmioQueryFormatCount - Remarks

An application can use this function to query the number of formats supported, and then call [mmioGetFormats](#) with the correct size of *pFormatInfoList* to obtain descriptive information in [MMFORMATINFO](#) structures.

---

## mmioQueryFormatCount - Related Functions

- [mmioGetFormats](#)
- 

## mmioQueryFormatCount - Example Code

The following code illustrates how to obtain the number of IOProcs matching the specified format data.



# mmioQueryHeaderLength Parameter - hmmio

**hmmio** (**HMMIO**) - input  
The open file handle returned by [mmioOpen](#).

# mmioQueryHeaderLength Parameter - plHeaderLength

**plHeaderLength** (**PLONG**) - output  
Pointer to a LONG. The size of the header in bytes is returned. If the MMIO\_TRANSLATEHEADER flag was set in the *ulTranslate* field of **MMIOINFO** on [mmioOpen](#), it is the size of one of the standard headers listed below. Otherwise, it is the size of a native (untranslated) header for this type of file.

# mmioQueryHeaderLength Parameter - ulReserved

**ulReserved** (**ULONG**) - input  
Reserved for future use and must be set to zero.

# mmioQueryHeaderLength Parameter - ulFlags

**ulFlags** (**ULONG**) - input  
Reserved for future use and must be set to zero.

# mmioQueryHeaderLength Return Value - rc

**rc** (**ULONG**) - returns  
Return codes indicating success or type of failure:

MMIO_SUCCESS	If the function succeeds, 0 is returned.
MMIO_ERROR	The specified file is not a media file format type.
MMIOERR_INVALID_PARAMETER	An invalid parameter was passed.
MMIOERR_INVALID_HANDLE	The handle passed was not valid.

MMIOERR\_SEEK\_FAILED

A seek operation prior to a write- or read-advance operation failed.

-----

## mmioQueryHeaderLength - Parameters

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

**piHeaderLength** ([PLONG](#)) - output

Pointer to a LONG. The size of the header in bytes is returned. If the MMIO\_TRANSLATEHEADER flag was set in the *ulTranslate* field of [MMIOINFO](#) on [mmioOpen](#), it is the size of one of the standard headers listed below. Otherwise, it is the size of a native (untranslated) header for this type of file.

**ulReserved** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

**ulFlags** ([ULONG](#)) - input

Reserved for future use and must be set to zero.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The specified file is not a media file format type.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_SEEK\_FAILED

A seek operation prior to a write- or read-advance operation failed.

-----

## mmioQueryHeaderLength - Remarks

The application calls `mmioQueryHeaderLength` first to determine the buffer size that is needed by [mmioGetHeader](#) to obtain header data. This is required because headers for different formats are variable in length.

The header is different for each Media Type. The currently defined values for each *ulMediaType* ([MMIOINFO](#) structure) follow:

MMIO\_MEDIATYPE\_IMAGE

The data represents a still image. Images use [MMIMAGEHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_AUDIO

The data represents digital audio. Digital-audio data streams use [MMAUDIOHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_MIDI

The data represents MIDI streams. MIDI data streams use [MMMIDIHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_DIGITALVIDEO

The data represents digital video. Digital video data streams use [MMVIDEOHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_MOVIE

The data represents a movie. Movie data uses [MMMIEHEADER](#) as the media structure.

---

## mmioQueryHeaderLength - Related Functions

- [mmioGetInfo](#)
- [mmioSetInfo](#)

---

## mmioQueryHeaderLength - Example Code

The following code illustrates how to determine the header size.

```
HMMIO  hmmio1;
LONG    lHeaderLength;
ULONG   ulReserved = 0L;
ULONG   ulFlags = 0L;
ULONG   rc;
...

rc = mmioQueryHeaderLength( hmmio1,
                           &lHeaderLength,
                           ulReserved,
                           ulFlags);

if (rc)
    /* error */
else
    ...
```

---

## mmioQueryHeaderLength - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioQueryIOProcModuleHandle

---

## mmioQueryIOProcModuleHandle - Syntax

This function provides the module handle of an I/O procedure's DLL. This handle must be used to retrieve resources from the DLL.

```
#define INCL_MMIOOS2
#include <os2.h>

PMMIOPROC    pIOProc;        /* Specifies entry point. */
PHMODULE     phmodIOProc;    /* Pointer. */
ULONG        rc;             /* Return codes. */

rc = mmioQueryIOProcModuleHandle(pIOProc,
    phmodIOProc);
```

---

## mmioQueryIOProcModuleHandle Parameter - pIOProc

**pIOProc** ([PMMIOPROC](#)) - input

Indicates a specific entry point of an I/O procedure for which the DLL module handle is to be retrieved. That is, the address of the DLL's entry point, directed to a PMMIOPROC.

---

## mmioQueryIOProcModuleHandle Parameter - phmodIOProc

**phmodIOProc** ([PHMODULE](#)) - output

Pointer to a PHMODULE. Returns the module handle to the DLL.

---

## mmioQueryIOProcModuleHandle Return Value - rc

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The function failed for a reason different from any of the following returns in this list.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_INTERNAL\_SYSTEM

An internal system error occurred.

---

## mmioQueryIOProcModuleHandle - Parameters

**pIOProc** ([PMMIOPROC](#)) - input

Indicates a specific entry point of an I/O procedure for which the DLL module handle is to be retrieved. That is, the address of the DLL's entry point, directed to a PMMIOPROC.

**phmodIOProc** ([PHMODULE](#)) - output

Pointer to a PHMODULE. Returns the module handle to the DLL.

**rc** ([ULONG](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The function failed for a reason different from any of the following returns in this list.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_INTERNAL\_SYSTEM

An internal system error occurred.

-----

## mmioQueryIOProcModuleHandle - Remarks

The mmioQueryIOProcModuleHandle function can only provide the handle to the DLL if it was loaded by MMIO from the MMPMMMIO.INI file.

-----

## mmioQueryIOProcModuleHandle - Related Functions

- [mmioGetInfo](#)
- [mmioSetInfo](#)

-----

## mmioQueryIOProcModuleHandle - Example Code

The following code illustrates how to obtain the module handle of an IOProc's DLL.

```
HMODULE hmodIOProc;
ULONG rc;
...

rc = mmioQueryIOProcModuleHandle((PMMIOPROC)&MyIOProc, &hmodIOProc)
if (rc)
    /* error */
else
    ...
```

The following is a procedure prototype for a standard I/O procedure call.

```
LONG APIENTRY MMIOPROC (PVOID pmmioinfo, USHORT usMsg,
                        LONG lParam1, LONG lParam2);
```

-----



# mmioQueryIOProcModuleHandle - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

---

## mmioRead

---

### mmioRead - Syntax

This function reads from a file opened by [mmioOpen](#) and updates the file position.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO    hmmio;        /* Open file handle. */
PCHAR    pchBuffer;    /* Buffer to read to. */
LONG     cBytes;        /* Number of bytes read. */
LONG     rc;           /* Return codes. */

rc = mmioRead(hmmio, pchBuffer, cBytes);
```

---

### mmioRead Parameter - hmmio

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

---

### mmioRead Parameter - pchBuffer

**pchBuffer** ([PCHAR](#)) - input

The buffer to read to.

---

# mmioRead Parameter - cBytes

**cBytes** ([LONG](#)) - input

The number of bytes to read from the file into the *pchBuffer* parameter.

---

## mmioRead Return Value - rc

**rc** ([LONG](#)) - returns

Returns the number of bytes actually read. If no more bytes can be read because the end of file has been reached, 0 is returned. If an error occurs, MMIO\_ERROR is returned. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_WRITE\_ONLY\_FILE

File not opened in Read mode.

MMIOERR\_READ\_FAILED

Unable to read; probable hardware error.

MMIOERR\_WRITE\_FAILED

Writing to a full buffer before trying to read the next buffer failed.

MMIOERR\_SEEK\_FAILED

Unable to seek; probable hardware error.

MMIOERR\_INVALID\_BUFFER\_LENGTH

The buffer length is invalid.

MMIOERR\_NO\_BUFFER\_ALLOCATED

Write operation expected a buffer but none was allocated.

---

## mmioRead - Parameters

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

**pchBuffer** ([PCHAR](#)) - input

The buffer to read to.

**cBytes** ([LONG](#)) - input

The number of bytes to read from the file into the *pchBuffer* parameter.

**rc** ([LONG](#)) - returns

Returns the number of bytes actually read. If no more bytes can be read because the end of file has been reached, 0 is returned. If an error occurs, MMIO\_ERROR is returned. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_WRITE\_ONLY\_FILE

File not opened in Read mode.

MMIOERR\_READ\_FAILED

Unable to read; probable hardware error.

MMIOERR\_WRITE\_FAILED

Writing to a full buffer before trying to read the next buffer failed.

MMIOERR\_SEEK\_FAILED

Unable to seek; probable hardware error.

MMIOERR\_INVALID\_BUFFER\_LENGTH  
The buffer length is invalid.

MMIOERR\_NO\_BUFFER\_ALLOCATED  
Write operation expected a buffer but none was allocated.

---

## mmioRead - Remarks

If the MMIO\_TRANSLATEDATA flag was in the *ulTranslate* field of the [MMIOINFO](#) structure when the file was opened, the data will be translated from its native encoding scheme to the encoding scheme of the standard presentation format for the media type. Data in the *pchBuffer* parameter is returned to the application in the standard presentation format.

With images, for example, the standard data presentation format is uncompressed OS/2 bit-map data. For audio, the standard presentation is PCM data. The data will conform to the definition found in the media header supplied by [mmioGetHeader](#).

---

## mmioRead - Related Functions

- [mmioClose](#)
- [mmioOpen](#)
- [mmioSeek](#)
- [mmioWrite](#)

---

## mmioRead - Example Code

The following code illustrates how to read from a file.

```
HMMIO hmmio1;
PCHAR pchBuffer;
LONG cBytes;
LONG lBytesRead;
...

lBytesRead = mmioRead(hmmio1, pchBuffer, cBytes);

if (lBytesRead < 0L)
    /* error */
else
    ...
```

---

## mmioRead - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)

[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioRemoveElement

---

### mmioRemoveElement - Syntax

This function removes a specific element from a compound file. mmioRemoveElement is a 32-bit function that is also provided as a 16-bit entry point.

```
#define INCL_MACHDR
#include <os2.h>

PSZ      pszFileElement; /* Pointer to file element name. */
ULONG    ulFlag;         /* Flags. */
ULONG    rc;              /* Return codes. */

rc = mmioRemoveElement(pszFileElement, ulFlag);
```

---

### mmioRemoveElement Parameter - pszFileElement

**pszFileElement** ([PSZ](#)) - input  
Pointer to a compound-file element name in the format: a:\path\file+element.

---

### mmioRemoveElement Parameter - ulFlag

**ulFlag** ([ULONG](#)) - input  
Specifies possible options. Contains 0 or the following flag:

**MMIO\_RE\_COMPACT**  
Compacts the compound file after removing the element. If no element is specified but this flag is set, the compound file will be compacted. If the element is specified but does not exist, no file compaction is done.

---

### mmioRemoveElement Return Value - rc

**rc** ([ULONG](#)) - returns

Returns MMIO\_SUCCESS if there was no error; otherwise it returns an error code.

MMIOERR\_CF\_ENTRY\_NOT\_FOUND

Element can not be found.

MMIOERR\_INVALID\_PARAMETER

No element name specified in the *pszFileElement* parameter and the MMIO\_RE\_COMPACT flag is not set.

-----

## mmioRemoveElement - Parameters

**pszFileElement** ([PSZ](#)) - input

Pointer to a compound-file element name in the format: a:\path\file+element.

**ulFlag** ([ULONG](#)) - input

Specifies possible options. Contains 0 or the following flag:

MMIO\_RE\_COMPACT

Compacts the compound file after removing the element. If no element is specified but this flag is set, the compound file will be compacted. If the element is specified but does not exist, no file compaction is done.

**rc** ([ULONG](#)) - returns

Returns MMIO\_SUCCESS if there was no error; otherwise it returns an error code.

MMIOERR\_CF\_ENTRY\_NOT\_FOUND

Element can not be found.

MMIOERR\_INVALID\_PARAMETER

No element name specified in the *pszFileElement* parameter and the MMIO\_RE\_COMPACT flag is not set.

-----

## mmioRemoveElement - Remarks

The mmioRemoveElement function is a high-level interface to remove an element from a compound file.

-----

## mmioRemoveElement - Related Functions

- [mmioFindElement](#)

-----

## mmioRemoveElement - Example Code

The following code illustrates how to remove an element from a compound file and compact it after it is removed.

```
ULONG ulReturnCode;  
ulReturnCode=mmioRemoveElement("test.bnd+element",MMIO_RE_COMPACT);  
if (ulReturnCode)  
... error  
... success
```

---

## mmioRemoveElement - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

---

## mmioSeek

---

### mmioSeek - Syntax

This function seeks within a file that was opened using [mmioOpen](#).

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO    hmmio;    /* Open file handle. */
LONG     lOffset;  /* Offset in bytes. */
LONG     lOrigin;  /* Origin of offset. */
LONG     rc;        /* New file position. */

rc = mmioSeek(hmmio, lOffset, lOrigin);
```

---

### mmioSeek Parameter - hmmio

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

---

### mmioSeek Parameter - lOffset

**lOffset** ([LONG](#)) - input

Specifies an offset, in bytes, to move the file position to.

-----

## mmioSeek Parameter - IOrigin

**IOrigin** ([LONG](#)) - input

Specifies how the *IOffset* parameter is interpreted:

SEEK\_SET

Seek to an absolute (bytes from the beginning of the file) seek position specified in the *IOffset* parameter. This is the default.

SEEK\_CUR

Seek to a relative (bytes from the current file position) seek position specified in the *IOffset* parameter.

SEEK\_END

Seek to *IOffset* bytes from the end of the file.

MMIO\_SEEK\_IFRAME

Seek to the nearest I-frame based on one of the previous flags. This flag can only be used with digital video only data tracks (or files).

-----

## mmioSeek Return Value - rc

**rc** ([LONG](#)) - returns

Returns the new file position (in bytes) from the beginning of the file. If an error occurs, MMIO\_ERROR is returned. A call to [mmioGetLastError](#) might return the following error:

MMIOERR\_SEEK\_FAILED

Probable hardware error.

-----

## mmioSeek - Parameters

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

**IOffset** ([LONG](#)) - input

Specifies an offset, in bytes, to move the file position to.

**IOrigin** ([LONG](#)) - input

Specifies how the *IOffset* parameter is interpreted:

SEEK\_SET

Seek to an absolute (bytes from the beginning of the file) seek position specified in the *IOffset* parameter. This is the default.

SEEK\_CUR

Seek to a relative (bytes from the current file position) seek position specified in the *IOffset* parameter.

SEEK\_END

Seek to *IOffset* bytes from the end of the file.

MMIO\_SEEK\_IFRAME

Seek to the nearest I-frame based on one of the previous flags. This flag can only be used with digital video only data tracks (or files).

**rc** ([LONG](#)) - returns

Returns the new file position (in bytes) from the beginning of the file. If an error occurs, MMIO\_ERROR is returned. A call to [mmioGetLastError](#) might return the following error:

MMIOERR\_SEEK\_FAILED

Probable hardware error.

-----

## mmioSeek - Remarks

Seeking past the end of the file does not result in an error; mmioSeek will return the offset of the new file position. Be careful when seeking past the end of the file. To determine where the end of a file is, call mmioSeek with the *lOffset* parameter equal to 0 and the *lOrigin* parameter equal to SEEK\_END.

**Note:** It is invalid to seek backwards (negative) from the beginning of the file.

In the case of a user-supplied memory (MEM) file, a SEEK\_END will seek from the end of the buffer, which might be different from the actual end of the data.

-----

## mmioSeek - Related Functions

- [mmioClose](#)
- [mmioOpen](#)
- [mmioRead](#)
- [mmioWrite](#)

-----

## mmioSeek - Example Code

The following code illustrates how to seek within a file.

```
HMMIO hmmio1;
LONG lOffset;
LONG lOrigin = 0L;
LONG lFilePosition;
...

lOffset = 0L;
lOrigin |= SEEK_END;

lFilePosition = mmioSeek(hmmio1, lOffset, lOrigin);

if (lFilePosition < 0L)
    /* error */
else
    ...
```

-----

## mmioSeek - Topics



Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

---

## mmioSendMessage

---

### mmioSendMessage - Syntax

This function sends a message to the I/O procedure associated with a file that was opened with [mmioOpen](#).

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO    hmmio;    /* Open file handle. */
USHORT    usMsg;    /* Message #. */
LONG      lParam1;  /* Message information. */
LONG      lParam2;  /* Message information. */
LONG      rc;       /* Return codes. */

rc = mmioSendMessage(hmmio, usMsg, lParam1,
                    lParam2);
```

---

### mmioSendMessage Parameter - hmmio

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

---

### mmioSendMessage Parameter - usMsg

**usMsg** ([USHORT](#)) - input

A message number. See [MMIO Messages](#).

---

### mmioSendMessage Parameter - lParam1

**IParam1** ([LONG](#)) - input

Specifies additional message information. If */Param1* for the particular MMIO message being sent is not a LONG value, cast the value into a LONG data type.

---

## mmioSendMessage Parameter - IParam2

**IParam2** ([LONG](#)) - input

Specifies additional message information. If */Param2* for the particular MMIO message being sent is not a LONG value, cast the value into a LONG data type.

---

## mmioSendMessage Return Value - rc

**rc** ([LONG](#)) - returns

The return code is specific to the message sent. This includes both successful and failed returns.

Regarding message-specific return codes, additional information may be contained in the *uiErrorRet* field of the [MMIOINFO](#) structure. This is accessed by calling [mmioGetLastError](#).

MMIO\_ERROR

The message cannot be routed to an IOProc. The handle might be invalid.

MMIOERR\_UNSUPPORTED\_MESSAGE

The I/O procedure does not support the message.

---

## mmioSendMessage - Parameters

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

**usMsg** ([USHORT](#)) - input

A message number. See [MMIO Messages](#).

**IParam1** ([LONG](#)) - input

Specifies additional message information. If */Param1* for the particular MMIO message being sent is not a LONG value, cast the value into a LONG data type.

**IParam2** ([LONG](#)) - input

Specifies additional message information. If */Param2* for the particular MMIO message being sent is not a LONG value, cast the value into a LONG data type.

**rc** ([LONG](#)) - returns

The return code is specific to the message sent. This includes both successful and failed returns.

Regarding message-specific return codes, additional information may be contained in the *uiErrorRet* field of the [MMIOINFO](#) structure. This is accessed by calling [mmioGetLastError](#).

MMIO\_ERROR

The message cannot be routed to an IOProc. The handle might be invalid.

MMIOERR\_UNSUPPORTED\_MESSAGE

The I/O procedure does not support the message.

-----

## mmioSendMessage - Remarks

An application can issue mmioSendMessage to send private messages to an installable I/O procedure. This function enables a program to call an I/O procedure directly (unlike system messages, which should be sent by the MMIO Manager). MMIOOS2.H in the \TOOLKIT\H subdirectory defines the identifier MMIOM\_USER so that you can create your own messages. The mmioSendMessage function requires that your custom messages be defined between the MMIOM\_USER and MMIOM\_USER\_END values defined in the MMIOOS2.H file.

-----

## mmioSendMessage - Related Functions

- [mmioInstallIOProc](#)

-----

## mmioSendMessage - Example Code

The following code illustrates how to send a message to the IOProc.

```
HMMIO hmmiol;  
USHORT usMsg;  
LONG lParam1 = 0L;  
LONG lParam2 = 0L;  
LONG rc;  
  
...  
  
usMsg = MMIOM_GETCF;  
  
rc = mmioSendMessage(hmmiol, usMsg, lParam1, lParam2);  
  
if (!rc)  
    /* error */  
else  
    ...
```

-----

## mmioSendMessage - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

---

# mmioSet

---

## mmioSet - Syntax

This function can be used to set or query various extended information. It can associate a CODEC with an I/O procedure, set the current track for multiple track files, set the playing speed of a digital video, and so forth.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO      hmmio;          /* Open file handle. */
PMMEXTENDINFO pUserExtendmminfo; /* Extended information. */
ULONG      ulFlags;        /* Flags. */
ULONG      rc;             /* Return codes. */

rc = mmioSet(hmmio, pUserExtendmminfo, ulFlags);
```

---

## mmioSet Parameter - hmmio

**hmmio** (**HMMIO**) - input  
The MMIO file handle returned by [mmioOpen](#).

---

## mmioSet Parameter - pUserExtendmminfo

**pUserExtendmminfo** (**PMMEXTENDINFO**) - input  
Pointer to the [MMEXTENDINFO](#) structure.

---

## mmioSet Parameter - ulFlags

**ulFlags** (**ULONG**) - input  
This parameter contains one of the following flags:

**Note:** To reference a track other than the default track with the QUERY and SET calls, the MMIO\_TRACK and MMIO\_CODEC\_ASSOC flags must be set at the same time the QUERY or SET is performed.

MMIO\_SET\_EXTENDEDINFO  
Set the extended information.

MMIO\_QUERY\_EXTENDEDINFO\_BASE  
Query only the information of the [MMEXTENDINFO](#) structure.

MMIO\_QUERY\_EXTENDEDINFO\_ALL  
Query all extended information including the CODEC associated information.

---

## mmioSet Return Value - rc

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure:

MMIO\_SUCCESS  
If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE  
Invalid MMIO handle was passed.

MMIOERR\_INVALID\_PARAMETER  
An invalid parameter was passed.

---

## mmioSet - Parameters

**hmmio** ([HMMIO](#)) - input  
The MMIO file handle returned by [mmioOpen](#).

**pUserExtendmminfo** ([PMMEXTENDINFO](#)) - input  
Pointer to the [MMEXTENDINFO](#) structure.

**ulFlags** ([ULONG](#)) - input  
This parameter contains one of the following flags:

**Note:** To reference a track other than the default track with the QUERY and SET calls, the MMIO\_TRACK and MMIO\_CODECS\_ASSOC flags must be set at the same time the QUERY or SET is performed.

MMIO\_SET\_EXTENDEDINFO  
Set the extended information.

MMIO\_QUERY\_EXTENDEDINFO\_BASE  
Query only the information of the [MMEXTENDINFO](#) structure.

MMIO\_QUERY\_EXTENDEDINFO\_ALL  
Query all extended information including the CODEC associated information.

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure:

MMIO\_SUCCESS  
If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE  
Invalid MMIO handle was passed.

MMIOERR\_INVALID\_PARAMETER  
An invalid parameter was passed.

---

## mmioSet - Remarks

If MMIO\_SET\_EXTENDEDINFO is set to associate a CODEC procedure with an open file, the *pCODECIniFileInfo* field of the [CODECASSOC](#) structure is used to identify each CODEC procedure installed in the initialization file. As a result of the set, the CODEC procedures are opened and each *pCodecOpen* structure is passed to its corresponding CODEC procedure.

On query, two levels of information can be returned. If MMIO\_QUERY\_EXTENDEDINFO\_BASE is set, only the [MMEXTENDINFO](#) structure is returned. The *ulNumCODECs* is the number of currently associated CODEC procedures. The *ulBufSize* field is the buffer size for the second level information. If the application decides to query the second level information, the MMIO\_QUERY\_EXTENDEDINFO\_ALL flag must be set and the *pUserExtendmminfo* parameter must point to a buffer with the size equal to the *ulBufSize* field of the [MMEXTENDINFO](#) structure.

This function associates a CODEC procedure with an MMIO handle. Typically, this function is used to provide CODEC information for a new file being created. When an existing movie file is opened, any necessary CODEC procedures are loaded by the I/O procedure automatically based on the compression type and subtype specified in the file's header. However, there might be a need to change the output format (for example, color depth) of a CODEC and this function can be used for that. The default color depth is set to the display mode color depth for files opened for reading (that is, playback of a movie file).

If this function is not issued, no data compression and decompression will be performed for MMIO\_READ and MMIO\_WRITE.

**Note:** for digital video files, all reads and writes are MULTITRACK\_READ and MULTITRACK\_WRITE.)

---

## mmioSet - Example Code

The following code illustrates how to set CODEC information for an opened file.

```
HMMIO hmmiol;
MMEXTENDINFO mmExtendInfo;
CODECASSOC codecAssoc;
CODECINIFILEINFO codecIniFileInfo;
ULONG ulFlags;
ULONG rc;
...

hmmiol = mmioOpen("MYFILE.SMV", &mmioInfo, MMIO_CREATE);
mmExtendInfo.ulStructLen = sizeof(MMEXTENDINFO);
mmExtendInfo.ulFlags = MMIO_CODEC_ASSOC;
mmExtendInfo.ulNumCODECs = 1;
mmExtendInfo.pCODECAssoc = &codecAssoc;
codecIniFileInfo.ulStructLen = sizeof(CODECINIFILEINFO);
codecIniFileInfo.fcc = FOURCC_MYPROC
codecIniFileInfo.ulCompressType = COMPRESSTYPE_MYPROC;
codecIniFileInfo.ulCompressSubType = COMPRESSSUBTYPE_MYPROC;
codecIniFileInfo.ulMediaType = MEDIATYPE_MYPROC;
codecIniFileInfo.ulCapsFlags = CODEC_DECOMPRESS;
codecIniFileInfo.szHWID = HWID_MYPROC;
codecAssoc.pCODECIniFileInfo = &codecIniFileInfo;
codecAssoc.pCodecOpen = NULL;

ulFlags = MMIO_SET_EXTENDEDINFO;
rc = mmioSet(hmmiol, mmExtendInfo, ulFlags);
if (rc)
    /* error */
else
```

---

## mmioSet - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Glossary](#)

---

## mmioSetBuffer

---

### mmioSetBuffer - Syntax

This function enables or disables buffered I/O, or changes the buffer or buffer size, for a file that was opened using [mmioOpen](#).

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO    hmmio;        /* Open file handle. */
PCHAR    pchBuffer;    /* Pointer. */
LONG     cBytes;        /* Buffer size. */
USHORT    usFlags;      /* Flags. */
USHORT    rc;           /* Return codes. */

rc = mmioSetBuffer(hmmio, pchBuffer, cBytes,
                  usFlags);
```

---

### mmioSetBuffer Parameter - hmmio

**hmmio** ([HMMIO](#)) - input  
The open file handle returned by [mmioOpen](#).

---

### mmioSetBuffer Parameter - pchBuffer

**pchBuffer** ([PCHAR](#)) - input  
A pointer to the caller-supplied buffer to use for buffered I/O. It can be NULL if the caller wants mmioSetBuffer to allocate the buffer, or wants buffered I/O to be disabled.

---

### mmioSetBuffer Parameter - cBytes

**cBytes** ([LONG](#)) - input

The size of the caller-supplied buffer, or (if *pchBuffer* is NULL) the size of the buffer that the caller wants mmioSetBuffer to allocate.

---

## mmioSetBuffer Parameter - usFlags

**usFlags** ([USHORT](#)) - input

Reserved for future use and must be set to zero.

---

## mmioSetBuffer Return Value - rc

**rc** ([USHORT](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not correct.

MMIOERR\_UNBUFFERED

Tried to disable a buffer already disabled.

MMIOERR\_INVALID\_BUFFER\_LENGTH

The buffer length is invalid.

MMIOERR\_CANNOTWRITE

The buffer could not be written to disk. It might be full.

MMIOERR\_READ\_FAILED

Set Buffer failed during a read operation.

MMIOERR\_SEEK\_FAILED

Set Buffer failed during a seek operation.

MMIOERR\_WRITE\_FAILED

Set Buffer failed during a write operation.

MMIOERR\_OUTOFMEMORY

A buffer was expected but not allocated.

---

## mmioSetBuffer - Parameters

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

**pchBuffer** ([PCHAR](#)) - input

A pointer to the caller-supplied buffer to use for buffered I/O. It can be NULL if the caller wants mmioSetBuffer to allocate the buffer, or wants buffered I/O to be disabled.



**cBytes** ([LONG](#)) - input

The size of the caller-supplied buffer, or (if *pchBuffer* is NULL) the size of the buffer that the caller wants mmioSetBuffer to allocate.

**usFlags** ([USHORT](#)) - input

Reserved for future use and must be set to zero.

**rc** ([USHORT](#)) - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIOERR\_INVALID\_HANDLE

The handle passed was not correct.

MMIOERR\_UNBUFFERED

Tried to disable a buffer already disabled.

MMIOERR\_INVALID\_BUFFER\_LENGTH

The buffer length is invalid.

MMIOERR\_CANNOTWRITE

The buffer could not be written to disk. It might be full.

MMIOERR\_READ\_FAILED

Set Buffer failed during a read operation.

MMIOERR\_SEEK\_FAILED

Set Buffer failed during a seek operation.

MMIOERR\_WRITE\_FAILED

Set Buffer failed during a write operation.

MMIOERR\_OUTOFMEMORY

A buffer was expected but not allocated.

-----

## mmioSetBuffer - Remarks

If the *pchBuffer* parameter is NULL and the *cchBuffer* field of [MMIOINFO](#) is 0, buffered I/O is disabled. If *pchBuffer* is NULL and *cchBuffer* is not 0 and buffering was *enabled* before mmioSetBuffer was called and the I/O buffer was allocated by [mmioOpen](#) or a previous call to mmioSetBuffer, then mmioSetBuffer reallocates the I/O buffer to be *cchBuffer* bytes in length. The contents of the buffer are not disturbed in this case (though if the buffer is shrunk, some data will be lost), unless the current file position is in part of the buffer that is truncated.

If *pchBuffer* is NULL and *cchBuffer* is not 0 and buffering was *disabled* before mmioSetBuffer was called, then mmioSetBuffer allocates an I/O buffer of *cchBuffer* bytes in length, and buffered I/O is enabled.

If *pchBuffer* is not NULL and *cchBuffer* is not 0, then *pchBuffer* is assumed to be a caller-provided I/O buffer of *cchBuffer* bytes in length, which is used for buffered I/O.

-----

## mmioSetBuffer - Related Functions

- [mmioFlush](#)

-----

## mmioSetBuffer - Example Code

The following code illustrates how to set the buffer.

```
HMMIO hmmiol;
LONG cchBuffer;
PCHAR pchBuffer;
USHORT usFlags = 0;
USHORT rc;

...

pchBuffer = NULL;
cchBuffer = 0L;

rc = mmioSetBuffer(hmmiol, pchBuffer, cchBuffer, usFlags);

if (rc)
    /* error */
else
    ...
```

---

## mmioSetBuffer - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioSetHeader

---

## mmioSetHeader - Syntax

This function sets attributes of the media in a file opened for writing by [mmioOpen](#). It issues an [MMIOM\\_SETHEADER](#) message to the I/O procedure. The specific header depends on the media type of the file and current track setting, in the case of multiple tracks. This header can be a raw header or a translated header.

This function does not change the current file position. Typically, [mmioGetHeader](#) is issued to obtain the attribute data and [mmioSetHeader](#) is issued to update it. [mmioSetHeader](#) can be issued independently of [mmioGetHeader](#), such as to create the header initially.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO    hmmio;           /* Open file handle. */
PVOID    pHeader;         /* Pointer to header structure. */
LONG     lHeaderLength;    /* Size of header structure. */
PLONG    plBytesWritten;   /* # of bytes written. */
ULONG    ulReserved;       /* Reserved. */
ULONG    ulFlags;          /* Reserved. */
```

```
ULONG    rc;                /* Return codes. */

rc = mmioSetHeader(hmmio, pHeader, lHeaderLength,
                  plBytesWritten, ulReserved, ulFlags);
```

---

## mmioSetHeader Parameter - hmmio

**hmmio** ([HMMIO](#)) - input  
The open file handle returned by [mmioOpen](#).

---

## mmioSetHeader Parameter - pHeader

**pHeader** ([PVOID](#)) - input  
Pointer to a header structure. This structure contains the data that is written to the header.

If the MMIO\_TRANSLATEHEADER flag was set on the [mmioOpen](#) in the *ulTranslate* field of the [MMIOINFO](#) structure on the [mmioOpen](#) function, then the header expected by the call is one associated with the standard presentation format for that particular multimedia data type (media type). Each media type (see the *ulMediaType* field of the [MMFORMATINFO](#) structure.) has a different standard presentation header.

The I/O procedure is expected to transpose the header from this standard format into the native format before writing the header to the file.

If MMIO\_NOTTRANSLATE was specified on the open (default case) then the header is in its native format. The currently defined values for each *ulMediaType* and their respective media structures are as follows:

MMIO\_MEDIATYPE\_IMAGE  
The data represents a still image. Images use the [MMIMAGEHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_AUDIO  
The data represents digital audio. Digital-audio data streams use [MMAUDIOHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_MIDI  
The data represents MIDI streams. MIDI data streams use [MMMIDIHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_DIGITALVIDEO  
The data represents digital video. Digital video data uses [MMVIDEOHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_MOVIE  
The data represents a movie. Movie data uses [MMMOTIEHEADER](#) as the media structure.

---

## mmioSetHeader Parameter - lHeaderLength

**lHeaderLength** ([LONG](#)) - input  
The size of the header structure in bytes.

# mmioSetHeader Parameter - plBytesWritten

**plBytesWritten** ([PLONG](#)) - in/out  
Returns the number of bytes written to the header structure.

---

# mmioSetHeader Parameter - ulReserved

**ulReserved** ([ULONG](#)) - input  
Reserved for future use and must be set to zero.

---

# mmioSetHeader Parameter - ulFlags

**ulFlags** ([ULONG](#)) - input  
Reserved for future use and must be set to zero.

---

# mmioSetHeader Return Value - rc

**rc** ([ULONG](#)) - returns  
Return codes indicating success or type of failure:

MMIO_SUCCESS	If the function succeeds, 0 is returned.
MMIO_ERROR	The specified file is not a media-file format type.
MMIOERR_INVALID_PARAMETER	An invalid parameter was passed.
MMIOERR_INVALID_HANDLE	The handle passed was not valid.
MMIOERR_SEEK_FAILED	A seek operation prior to a write- or read-advance operation failed.

---

# mmioSetHeader - Parameters

**hmmio** ([HMMIO](#)) - input

The open file handle returned by [mmioOpen](#).

**pHeader (PVOID)** - input

Pointer to a header structure. This structure contains the data that is written to the header.

If the MMIO\_TRANSLATEHEADER flag was set on the [mmioOpen](#) in the *ulTranslate* field of the [MMIOINFO](#) structure on the [mmioOpen](#) function, then the header expected by the call is one associated with the standard presentation format for that particular multimedia data type (media type). Each media type (see the *ulMediaType* field of the [MMFORMATINFO](#) structure.) has a different standard presentation header.

The I/O procedure is expected to transpose the header from this standard format into the native format before writing the header to the file.

If MMIO\_NOTTRANSLATE was specified on the open (default case) then the header is in its native format. The currently defined values for each *ulMediaType* and their respective media structures are as follows:

MMIO\_MEDIATYPE\_IMAGE

The data represents a still image. Images use the [MMIMAGEHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_AUDIO

The data represents digital audio. Digital-audio data streams use [MMAUDIOHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_MIDI

The data represents MIDI streams. MIDI data streams use [MMMIDIHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_DIGITALVIDEO

The data represents digital video. Digital video data uses [MMVIDEOHEADER](#) as the media structure.

MMIO\_MEDIATYPE\_MOVIE

The data represents a movie. Movie data uses [MMMIEHEADER](#) as the media structure.

**lHeaderLength (LONG)** - input

The size of the header structure in bytes.

**plBytesWritten (PLONG)** - in/out

Returns the number of bytes written to the header structure.

**ulReserved (ULONG)** - input

Reserved for future use and must be set to zero.

**ulFlags (ULONG)** - input

Reserved for future use and must be set to zero.

**rc (ULONG)** - returns

Return codes indicating success or type of failure:

MMIO\_SUCCESS

If the function succeeds, 0 is returned.

MMIO\_ERROR

The specified file is not a media-file format type.

MMIOERR\_INVALID\_PARAMETER

An invalid parameter was passed.

MMIOERR\_INVALID\_HANDLE

The handle passed was not valid.

MMIOERR\_SEEK\_FAILED

A seek operation prior to a write- or read-advance operation failed.

-----

## mmioSetHeader - Remarks

The contents of the header *must* represent the structure that is expected by the I/O procedure. It does not represent the manner in which

the data will be saved by the I/O procedure, because the I/O procedure might translate the data in some manner. The *pBytesWritten* parameter value might differ from the actual number of bytes written to the file in the case of translations.

This function can be used in conjunction with [mmioSet](#) to set (write) a specific track header into a multiple track movie file.

---

## mmioSetHeader - Related Functions

- [mmioGetHeader](#)
- [mmioQueryHeaderLength](#)

---

## mmioSetHeader - Example Code

The following code illustrates how to update header attributes.

```
HMMIO hmmiol;
MMAUDIOHEADER mmAudioHeader;
LONG lBytesWritten;
ULONG ulReserved = 0L;
ULONG ulFlags = 0L;
ULONG rc;
...

memset( &mmAudioHeader, '\0', sizeof(MMAUDIOHEADER));
...

rc = mmioSetHeader( hmmiol,
                    (PVOID)&mmAudioHeader,
                    (LONG)sizeof(MMAUDIOHEADER),
                    &lBytesWritten,
                    ulReserved,
                    ulFlags);

if (rc)
    /* error */
else
    ...
```

---

## mmioSetHeader - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioSetInfo

---

## mmioSetInfo - Syntax

This function updates information on a file I/O buffer of a file opened for buffered I/O.

```
#define INCL_MMIOOS2
#include <os2.h>

HMMIO      hmmio;      /* Open file handle. */
PMMIOINFO  pmmioinfo;  /* Buffer. */
USHORT     usFlags;     /* Reserved. */
USHORT     rc;          /* Return codes. */

rc = mmioSetInfo(hmmio, pmmioinfo, usFlags);
```

---

### mmioSetInfo Parameter - hmmio

**hmmio** ([HMMIO](#)) - input  
The open file handle returned by [mmioOpen](#).

---

### mmioSetInfo Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#)) - input  
The caller-allocated [MMIOINFO](#) buffer that was filled with information by [mmioGetInfo](#).

---

### mmioSetInfo Parameter - usFlags

**usFlags** ([USHORT](#)) - input  
Reserved for future use and must be set to zero.

---

### mmioSetInfo Return Value - rc

**rc** ([USHORT](#)) - returns  
Return codes indicating success or type of failure:

MMIO\_SUCCESS  
If the function succeeds, 0 is returned.

MMIOERR\_UNBUFFERED  
Tried to disable a buffer that was already disabled.

MMIOERR\_INVALID\_HANDLE  
The handle passed was not correct.

MMIOERR\_INVALID\_PARAMETER  
An incorrect parameter was passed.

---

## mmioSetInfo - Parameters

**hmmio** ([HMMIO](#)) - input  
The open file handle returned by [mmioOpen](#).

**pmmioinfo** ([PMMIOINFO](#)) - input  
The caller-allocated [MMIOINFO](#) buffer that was filled with information by [mmioGetInfo](#).

**usFlags** ([USHORT](#)) - input  
Reserved for future use and must be set to zero.

**rc** ([USHORT](#)) - returns  
Return codes indicating success or type of failure:

MMIO\_SUCCESS  
If the function succeeds, 0 is returned.

MMIOERR\_UNBUFFERED  
Tried to disable a buffer that was already disabled.

MMIOERR\_INVALID\_HANDLE  
The handle passed was not correct.

MMIOERR\_INVALID\_PARAMETER  
An incorrect parameter was passed.

---

## mmioSetInfo - Remarks

If using buffered I/O, make sure you set the MMIO\_DIRTY flag in the *ulFlags* field of [MMIOINFO](#) (before calling `mmioSetInfo`), if you have written to the buffer. Otherwise, the contents of the buffer will not be written to disk.

---

## mmioSetInfo - Related Functions

- [mmioAdvance](#)
  - [mmioGetInfo](#)
- 

## mmioSetInfo - Example Code



The following code illustrates how to update open file information.

```
HMMIO hmmiol;
MMIOINFO mmioinfo;
USHORT usFlags = 0;
USHORT rc;
...

memset( &mmioinfo, '\0', sizeof(MMIOINFO) );
rc = mmioGetInfo( hmmiol, &mmioinfo, usFlags);
if (rc)
    /* error */
else
    /* do some low-level I/O */
    ...

rc = mmioSetInfo( hmmiol, &mmioinfo, usFlags);

if (rc)
    /* error */
else
    ...
```

---

## mmioSetInfo - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioStringToFOURCC

---

## mmioStringToFOURCC - Syntax

This function converts a null-terminated string to a four-character code (FOURCC).

```
#define INCL_MMIOOS2
#include <os2.h>

PSZ      pszString; /* String to convert. */
USHORT   usFlags;   /* Flags. */
FOURCC   rc;        /* Return codes. */

rc = mmioStringToFOURCC(pszString, usFlags);
```

---

## mmioStringToFOURCC Parameter - pszString

**pszString** (**PSZ**) - input  
The string to convert to a FOURCC.

---

## mmioStringToFOURCC Parameter - usFlags

**usFlags** (**USHORT**) - input  
Contains 0 or the following flag:

MMIO\_TOUPPER  
All the characters in the four-character code are converted to uppercase.

---

## mmioStringToFOURCC Return Value - rc

**rc** (**FOURCC**) - returns  
Returns the four-character code.

---

## mmioStringToFOURCC - Parameters

**pszString** (**PSZ**) - input  
The string to convert to a FOURCC.

**usFlags** (**USHORT**) - input  
Contains 0 or the following flag:

MMIO\_TOUPPER  
All the characters in the four-character code are converted to uppercase.

**rc** (**FOURCC**) - returns  
Returns the four-character code.

---

## mmioStringToFOURCC - Remarks

This function does not check to see if the *pszString* parameter follows any conventions regarding which characters to include in a FOURCC code. The string is simply copied to a FOURCC code, and padded with blanks to the right or truncated to four characters, as required.

---

## mmioStringToFOURCC - Related Functions

- [mmioAscend](#)
  - [mmioCreateChunk](#)
  - [mmioDescend](#)
  - [mmioFOURCC](#)
- 

## mmioStringToFOURCC - Example Code

The following code illustrates how to convert a string to a four-character code.

```
FOURCC fcc;
...

fcc = mmioStringToFOURCC( "IMG", MMIO_TOUPPER );

if (!fcc)
    /* error */
else
    ...
```

---

## mmioStringToFOURCC - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## mmioWrite

---

## mmioWrite - Syntax

This function writes to a file that was opened using [mmioOpen](#).

```
#define INCL_MMIOOS2
#include <os2.h>
```

```
HMMIO    hmmio;        /* Open file handle. */
PCHAR    pchBuffer;    /* Buffer to write from. */
LONG     cBytes;        /* Number of bytes. */
LONG     rc;            /* Number of bytes actually written. */

rc = mmioWrite(hmmio, pchBuffer, cBytes);
```

---

## mmioWrite Parameter - hmmio

**hmmio** ([HMMIO](#)) - input  
The open file handle returned by [mmioOpen](#).

---

## mmioWrite Parameter - pchBuffer

**pchBuffer** ([PCHAR](#)) - input  
The buffer to write from.

---

## mmioWrite Parameter - cBytes

**cBytes** ([LONG](#)) - input  
The number of bytes to write from the *pchBuffer* parameter buffer to the file.

---

## mmioWrite Return Value - rc

**rc** ([LONG](#)) - returns  
Returns the number of bytes actually written. If an error occurs, MMIO\_ERROR is returned. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_READ\_ONLY\_FILE  
File not opened in WRITE mode.

MMIOERR\_INVALID\_HANDLE  
Invalid handle specified.

MMIOERR\_WRITE\_FAILED  
Unable to write; probable hardware error.

MMIOERR\_SEEK\_FAILED  
Unable to seek; probable hardware error.

MMIOERR\_READ\_FAILED  
Unable to read; probable hardware error.

MMIOERR\_INVALID\_BUFFER\_LENGTH  
The buffer length is invalid.

MMIOERR\_NO\_BUFFER\_ALLOCATED  
A buffer was expected but none was found.

MMIOERR\_CANNOTWRITE  
The target media has no space available.

---

## mmioWrite - Parameters

**hmmio** ([HMMIO](#)) - input  
The open file handle returned by [mmioOpen](#).

**pchBuffer** ([PCHAR](#)) - input  
The buffer to write from.

**cBytes** ([LONG](#)) - input  
The number of bytes to write from the *pchBuffer* parameter buffer to the file.

**rc** ([LONG](#)) - returns  
Returns the number of bytes actually written. If an error occurs, MMIO\_ERROR is returned. A call to [mmioGetLastError](#) might return one of the following errors:

MMIOERR\_READ\_ONLY\_FILE  
File not opened in WRITE mode.

MMIOERR\_INVALID\_HANDLE  
Invalid handle specified.

MMIOERR\_WRITE\_FAILED  
Unable to write; probable hardware error.

MMIOERR\_SEEK\_FAILED  
Unable to seek; probable hardware error.

MMIOERR\_READ\_FAILED  
Unable to read; probable hardware error.

MMIOERR\_INVALID\_BUFFER\_LENGTH  
The buffer length is invalid.

MMIOERR\_NO\_BUFFER\_ALLOCATED  
A buffer was expected but none was found.

MMIOERR\_CANNOTWRITE  
The target media has no space available.

---

## mmioWrite - Remarks

For a memory file (MEM) that cannot expand, mmioWrite returns the number of bytes written. This might be fewer than requested if the end of file (EOF) was encountered prematurely. If the logical file pointer was past the EOF when the write operation was initiated, MMIO\_ERROR is returned. If a pointer is at the EOF, a 0 is returned indicating no bytes were written. If the file can expand, it will do so and write the number of bytes requested, even if the logical file pointer was past the EOF when the write operation was initiated.

**Note:** User buffers cannot be expanded, but system-allocated buffers can be expanded.

Elements of a compound file will behave similar to the way a memory file does. The key to whether an element can be expanded is if the element is opened with the MMIO\_APPEND flag set.

If the MMIO\_TRANSLATEDATA flag was sent when the file was opened, the data is expected in the standard presentation format for the specific media type. The I/O procedure will translate the data from the standard presentation format into the I/O procedure format-specific representation before writing to the file. With images, for example, the standard data presentation format is uncompressed OS/2 2.0 bit-map data. For audio, the standard representation is uncompressed PCM data. The data will conform to the definition found in the media header supplied by [mmioSetHeader](#).

---

## mmioWrite - Related Functions

- [mmioClose](#)
- [mmioGetLastError](#)
- [mmioOpen](#)
- [mmioRead](#)
- [mmioSeek](#)

---

## mmioWrite - Example Code

The following code illustrates how to write to a file.

```
HMMIO hmmiol;  
PCHAR pchBuffer;  
LONG cBytes;  
LONG lBytesWritten;  
...  
  
lBytesWritten = mmioWrite(hmmiol, pchBuffer, cBytes);  
  
if (lBytesWritten < 0L)  
    /* error */  
else  
    ...
```

---

## mmioWrite - Topics

Select an item:

[Syntax](#)  
[Parameters](#)  
[Returns](#)  
[Remarks](#)  
[Example Code](#)  
[Related Functions](#)  
[Glossary](#)

---

## MMIO Messages

The MMIO messages are sent to an I/O procedure as a result of an MMIO function call. For example, when an application calls [mmioOpen](#),

the [MMIOM\\_OPEN](#) message is sent to an I/O procedure. An application can also send the MMIO messages to an I/O procedure by issuing [mmioSendMessage](#) or by directly calling the I/O procedure.

The [mmioSendMessage](#) function should be used only to pass user-defined messages, or messages not automatically generated by an MMIO function, to the I/O procedure of the user's application.

When an application issues [mmioSendMessage](#), it passes an *hmmio* parameter, which MMIO converts to an [MMIOINFO](#) block before sending the parameter to the I/O procedure. When an application calls an I/O procedure directly, a *pmmioinfo* parameter is passed to the I/O procedure instead of an *hmmio*. The following is a function prototype for a direct I/O procedure call:

```
LONG APIENTRY MMIOPROC (PVOID pmmioinfo, USHORT usMsg,
                        LONG lParam1, LONG lParam2);
```

Message	Description
<a href="#">MMIOM_BEGININSERT</a>	Requests that all subsequent Writes insert data at the current seek position.
<a href="#">MMIOM_BEGINRECORD</a>	Requests all subsequent Writes be considered one logical unit by an UNDO or REDO.
<a href="#">MMIOM_BEGINSTREAM</a>	Sent before the first <a href="#">mmioRead</a> or <a href="#">mmioWrite</a> to start the stream at the optimum rate for the file.
<a href="#">MMIOM_CLEAR</a>	Requests that a specified range be deleted from a file.
<a href="#">MMIOM_CLOSE</a>	Requests that the file be closed.
<a href="#">MMIOM_COPY</a>	Requests that a specified range be copied to the clipboard.
<a href="#">MMIOM_CUT</a>	Requests that a specified range be copied to the clipboard and then deleted.
<a href="#">MMIOM_ENDSTREAM</a>	Sent after the last <a href="#">mmioRead</a> or <a href="#">mmioWrite</a> to end the stream.
<a href="#">MMIOM_DELETE</a>	Requests that information be removed from a file.
<a href="#">MMIOM_ENDINSERT</a>	Requests that all subsequent Writes overwrite data at the current seek position.
<a href="#">MMIOM_ENDRECORD</a>	Indicates that the logical record operation has ended and data structures should be updated, if necessary.
<a href="#">MMIOM_GETCF</a>	Obtains the handle ( <i>hmmcf</i> ) of the RIFF compound file.
<a href="#">MMIOM_GETCFENTRY</a>	Requests a CTOC entry for an element of a RIFF compound file.
<a href="#">MMIOM_GETFORMATINFO</a>	Requests that the IOProc return an <a href="#">MMFORMATINFO</a> structure.
<a href="#">MMIOM_GETFORMATNAME</a>	Requests the format name for the IOProc.
<a href="#">MMIOM_GETHEADER</a>	Requests that the IOProc return media-specific information.
<a href="#">MMIOM_IDENTIFYFILE</a>	Attempts to determine if any IOProc can process a file.
<a href="#">MMIOM_MULTITRACKREAD</a>	Requests that data be read from a movie file.

<code>MMIOM_MULTITRACKWRITE</code>	Requests that data be written to a movie file.
<code>MMIOM_OPEN</code>	Requests that a file be opened or deleted.
<code>MMIOM_PASTE</code>	Requests that data from the clipboard be inserted into a file.
<code>MMIOM_QUERYHEADERLENGTH</code>	Requests that the IOProc return the size the header.
<code>MMIOM_QUERYIMAGE</code>	Requests that the IOProc return the currently selected image index in the image file.
<code>MMIOM_QUERYIMAGECOUNT</code>	Requests that the IOProc return the number of images stored in the image file.
<code>MMIOM_READ</code>	Requests that bytes be read from an open file.
<code>MMIOM_REDO</code>	Requests that the last logical action which was undone be redone.
<code>MMIOM_SAVE</code>	Requests temporary changes in a file.
<code>MMIOM_SEEK</code>	Requests that the current file position be moved.
<code>MMIOM_SEEKBYTIME</code>	Requests that the file position be moved relative to some unit of time.
<code>MMIOM_SET</code>	Requests that extended file information be set or queried.
<code>MMIOM_SETHEADER</code>	Requests that the IOProc use media-specific information when writing or accepting data.
<code>MMIOM_SETIMAGE</code>	Selects a new image index in the image file.
<code>MMIOM_STATUS</code>	Used to pass appropriate <code>MCI_STATUS</code> requests to an IOProc.
<code>MMIOM_TEMPCHANGE</code>	Requests that all subsequent Writes to the media be treated as temporary changes.
<code>MMIOM_UNDO</code>	Requests that the last logical action, either a delete, insert, undo, or redo, be undone.
<code>MMIOM_WINMSG</code>	Allows an application or an MCD to pass PM messages from a window procedure to an IOProc.
<code>MMIOM_WRITE</code>	Requests that the bytes be written to an open file.

The MMIO messages indicate, to the I/O procedure, the type of MMIO operation to be performed. I/O procedures can be called to process files that might or might not use the RIFF format standard.

**Note:** The compound-file messages must not be used while creating or appending to the compound file itself.

---

## MMIOM\_BEGININSERT



-----

## MMIOM\_BEGININSERT Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

-----

## MMIOM\_BEGININSERT Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_BEGININSERT.

-----

## MMIOM\_BEGININSERT Parameter - IParam1

**IParam1** ([LONG](#))  
This parameter is not used.

-----

## MMIOM\_BEGININSERT Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

-----

## MMIOM\_BEGININSERT Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The file is in insert mode.
MMIO_ERROR	An error code is returned.

-----

# MMIOM\_BEGININSERT - Description

This message is sent to an I/O procedure to request that all subsequent [mmioWrite](#) calls insert data at the current seek position rather than overwriting the data.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_BEGININSERT.

**IParam1** ([LONG](#))  
This parameter is not used.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The file is in insert mode.
MMIO_ERROR	An error code is returned.

---

## MMIOM\_BEGININSERT - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Glossary](#)

---

## MMIOM\_BEGINRECORD

---

### MMIOM\_BEGINRECORD Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_BEGINRECORD Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_BEGINRECORD.

-----

## MMIOM\_BEGINRECORD Parameter - IParam1

**IParam1** ([LONG](#))  
This parameter is not used.

-----

## MMIOM\_BEGINRECORD Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

-----

## MMIOM\_BEGINRECORD Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

**MMIO\_SUCCESS**  
The request was successful.

**MMIO\_ERROR**  
An error code is returned.

-----

## MMIOM\_BEGINRECORD - Description

This message is sent to an I/O procedure to request that all subsequent [mmioWrite](#) be considered one logical unit by an [MMIOM\\_UNDO](#) or [MMIOM\\_REDO](#) message.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_BEGINRECORD.

**IParam1** ([LONG](#))  
This parameter is not used.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO\_SUCCESS  
The request was successful.

MMIO\_ERROR  
An error code is returned.

---

## MMIOM\_BEGINRECORD - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Glossary](#)

---

## MMIOM\_BEGINSTREAM

---

## MMIOM\_BEGINSTREAM Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_BEGINSTREAM Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_BEGINSTREAM.

---

## MMIOM\_BEGINSTREAM Parameter - IParam1

**IParam1** ([LONG](#))  
Contains one of the following values:  
  
STREAM\_READ

Read stream from server to client.

STREAM\_WRITE

Write stream from client to server.

-----

## MMIOM\_BEGINSTREAM Parameter - pParam2

**pParam2** ([PQOSInfo](#))

A pointer to a [QOSInfo](#) structure. This structure contains a variable length array of [QOS](#) structures. Each [QOS](#) structure contains one quality of service parameter (specified in the */QOSParamId* field of the [QOS](#) structure) which is of one of the following:

SERVICE\_REQUEST

This requests the type of service required for this stream. The */QOSValue* field of the [QOS](#) structure contains the type of service: GUARANTEED, DONTCARE, or DONTRESERVE.

MAX\_EE\_JITTER

The number of stream buffers for handling jitter. Buffers needed to store a single unit of data are separate. The */QOSValue* field of the [QOS](#) structure contains the number of buffers.

MAX\_DATA\_RATE

Maximum data rate in bytes per second. The */QOSValue* field of the [QOS](#) structure contains this information.

AVG\_DATA\_RATE

Average data rate in bytes per second. The */QOSValue* field of the [QOS](#) structure contains this information.

-----

## MMIOM\_BEGINSTREAM Return Value - rc

**rc** ([ULONG](#))

Return codes indicating success or failure:

MMIO\_SUCCESS

The file is in streaming mode.

MMIOERR\_UNSUPPORTED\_MESSAGE

IOProc does not support this message.

MMIOERR\_QOSUNAVAILABLE

Quality of service is unavailable.

MMIO\_ERROR

Streaming was unsuccessful due to other errors.

-----

## MMIOM\_BEGINSTREAM - Description

This message is sent before first [mmioRead](#) or [mmioWrite](#) streamed at optimum rate to the file.

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

**usMsg** (**USHORT**)

Set to MMIOM\_BEGINSTREAM.

**IPParam1** (**LONG**)

Contains one of the following values:

STREAM\_READ

Read stream from server to client.

STREAM\_WRITE

Write stream from client to server.

**pParam2** (**PQOSInfo**)

A pointer to a **QOSInfo** structure. This structure contains a variable length array of **QOS** structures. Each **QOS** structure contains one quality of service parameter (specified in the *IQOSParamId* field of the **QOS** structure) which is one of the following:

SERVICE\_REQUEST

This requests the type of service required for this stream. The *IQOSValue* field of the **QOS** structure contains the type of service: GUARANTEED, DONTCARE, or DONTRESERVE.

MAX\_EE\_JITTER

The number of stream buffers for handling jitter. Buffers needed to store a single unit of data are separate. The *IQOSValue* field of the **QOS** structure contains the number of buffers.

MAX\_DATA\_RATE

Maximum data rate in bytes per second. The *IQOSValue* field of the **QOS** structure contains this information.

AVG\_DATA\_RATE

Average data rate in bytes per second. The *IQOSValue* field of the **QOS** structure contains this information.

**rc** (**ULONG**)

Return codes indicating success or failure:

MMIO\_SUCCESS

The file is in streaming mode.

MMIOERR\_UNSUPPORTED\_MESSAGE

IOProc does not support this message.

MMIOERR\_QOSUNAVAILABLE

Quality of service is unavailable.

MMIO\_ERROR

Streaming was unsuccessful due to other errors.

-----

## MMIOM\_BEGINSTREAM - Remarks

Some of the quality of service parameters are required on each call, if one of these is missing, an MMIOERR\_UNSUPPORTED\_MESSAGE error is generated.

The QOS parameters are kept as a list to allow future addition of quality parameters. "Quality of service" is being explored by the research community. With newer applications, changes to QOS parameters may be required. For each of the QOS parameters, the *IQOSValue* is one of the following values:

GUARANTEED

The application requests guaranteed service and if requested QOS is unavailable, the connection is not made.

DONTCARE

The applications requests the given QOS, but if it is unavailable, a connection may be made without quality of service reservation.

DONTRESERVE

The application does not want guarantees, (same as no message). The remaining fields are meaningless and not examined.

-----

# MMIOM\_BEGINSTREAM - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Remarks](#)  
[Glossary](#)

---

## MMIOM\_CLEAR

---

### MMIOM\_CLEAR Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_CLEAR Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_CLEAR.

---

### MMIOM\_CLEAR Parameter - IParam1

**IParam1** ([LONG](#))  
A pointer to an [MMIO\\_EDIT\\_PARMS](#) structure.

---

### MMIOM\_CLEAR Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

# MMIOM\_CLEAR Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The request was successful.
MMIO_ERROR	An error code is returned.

-----

## MMIOM\_CLEAR - Description

This message is sent to an I/O procedure to request that a specified range be deleted from a file. The clipboard is not used for this operation.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_CLEAR.

**IParam1** ([LONG](#))  
A pointer to an [MMIO\\_EDIT\\_PARMS](#) structure.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The request was successful.
MMIO_ERROR	An error code is returned.

-----

## MMIOM\_CLEAR - Remarks

The media position will be at the position corresponding to the *ulStartTime* field of the [MMIO\\_EDIT\\_PARMS](#) structure.

The *ulDuration* field cannot be zero.

-----

## MMIOM\_CLEAR - Topics

Select an item:



[Description](#)  
[Returns](#)  
[Remarks](#)  
[Glossary](#)

---

# MMIOM\_CLOSE

---

## MMIOM\_CLOSE Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_CLOSE Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_CLOSE.

---

## MMIOM\_CLOSE Parameter - IParam1

**IParam1** ([LONG](#))  
The *usFlags* parameter of [mmioClose](#).

---

## MMIOM\_CLOSE Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_CLOSE Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The specified file was closed successfully.
MMIO_ERROR	An error code is returned.
MMIO_WARNING	The file was closed, but the IOProc expected additional data.

## MMIOM\_CLOSE - Description

This message is sent to an IOProc by [mmioClose](#) to request that a file be closed.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_CLOSE.

**IParam1** ([LONG](#))  
The *usFlags* parameter of [mmioClose](#).

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The specified file was closed successfully.
MMIO_ERROR	An error code is returned.
MMIO_WARNING	The file was closed, but the IOProc expected additional data.

## MMIOM\_CLOSE - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Glossary](#)

## MMIOM\_COMPRESS

-----

## MMIOM\_COMPRESS Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

-----

## MMIOM\_COMPRESS Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_COMPRESS.

-----

## MMIOM\_COMPRESS Parameter - IParam1

**IParam1** ([LONG](#))  
Pointer to the [MMCOMPRESS](#) structure.

-----

## MMIOM\_COMPRESS Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

-----

## MMIOM\_COMPRESS Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

<a href="#">MMIO_SUCCESS</a>	The data is decompressed successfully.
<a href="#">MMIO_ERROR</a>	An error code is returned.

-----

## MMIOM\_COMPRESS - Description

This message is sent to an I/O procedure to compress the uncompressed data in the buffer.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_COMPRESS.

**IParam1** ([LONG](#))  
Pointer to the [MMCOMPRESS](#) structure.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

<a href="#">MMIO_SUCCESS</a>	The data is decompressed successfully.
<a href="#">MMIO_ERROR</a>	An error code is returned.

---

## MMIOM\_COMPRESS - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Glossary](#)

---

## MMIOM\_COPY

---

## MMIOM\_COPY Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_COPY Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_COPY.

---

## MMIOM\_COPY Parameter - IParam1

**IParam1** ([LONG](#))  
A pointer to an [MMIO\\_EDIT\\_PARMS](#) structure.

---

## MMIOM\_COPY Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_COPY Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The request was successful.
MMIO_ERROR	An error code is returned.

---

## MMIOM\_COPY - Description

This message is sent to an I/O procedure to request that a specified range be copied to the clipboard.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_COPY.

**IParam1** ([LONG](#))  
A pointer to an [MMIO\\_EDIT\\_PARMS](#) structure.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))

Return codes indicating success or failure:

MMIO_SUCCESS	The request was successful.
MMIO_ERROR	An error code is returned.

---

## MMIOM\_COPY - Remarks

If data is already in the clipboard, it is overwritten with this message call. The media position is not changed by this operation.

The *ulDuration* field of the [MMIO\\_EDIT\\_PARMS](#) structure passed in the *lParam1* parameter cannot be set to 0.

---

## MMIOM\_COPY - Topics

Select an item:

- [Description](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

---

## MMIOM\_CUT

---

## MMIOM\_CUT Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_CUT Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_CUT.

---

## MMIOM\_CUT Parameter - lParam1

**IParam1** ([LONG](#))  
A pointer to an [MMIO\\_EDIT\\_PARMS](#) structure.

-----

## MMIOM\_CUT Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

-----

## MMIOM\_CUT Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

<a href="#">MMIO_SUCCESS</a>	The request was successful.
<a href="#">MMIO_ERROR</a>	An error code is returned.

-----

## MMIOM\_CUT - Description

This message is sent to an I/O procedure to request the specified range is copied to the clipboard and then deleted. It performs as if an [MMIOM\\_COPY](#) operation immediately followed by a MMIOM\_CUT operation were requested.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_CUT.

**IParam1** ([LONG](#))  
A pointer to an [MMIO\\_EDIT\\_PARMS](#) structure.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

<a href="#">MMIO_SUCCESS</a>	The request was successful.
<a href="#">MMIO_ERROR</a>	An error code is returned.

-----

# MMIOM\_CUT - Remarks

If data is already in the clipboard, it is overwritten with this message call. The media position will be at the position corresponding to the *ulStartTime* field of the [MMIO\\_EDIT\\_PARMS](#) structure passed in the *lParam1* parameter.

The *ulDuration* field of the [MMIO\\_EDIT\\_PARMS](#) structure cannot be set to 0.

-----

# MMIOM\_CUT - Topics

- Select an item:
- [Description](#)
  - [Returns](#)
  - [Remarks](#)
  - [Glossary](#)

-----

# MMIOM\_DECOMPRESS

-----

## MMIOM\_DECOMPRESS Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to the [MMIOINFO](#) data structure.

-----

## MMIOM\_DECOMPRESS Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_DECOMPRESS.

-----

## MMIOM\_DECOMPRESS Parameter - lParam1

**lParam1** ([LONG](#))  
Pointer to the [MMDECOMPRESS](#) structure.



---

# MMIOM\_DECOMPRESS Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

# MMIOM\_DECOMPRESS Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The data is decompressed successfully.
MMIO_ERROR	An error code is returned.

---

# MMIOM\_DECOMPRESS - Description

This message is sent to the IOProc to decompress the compressed data in the buffer.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to the [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_DECOMPRESS.

**IParam1** ([LONG](#))  
Pointer to the [MMDECOMPRESS](#) structure.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The data is decompressed successfully.
MMIO_ERROR	An error code is returned.

---

# MMIOM\_DECOMPRESS - Topics

Select an item:

[Description](#)  
[Returns](#)  
[Glossary](#)

---

# MMIOM\_DELETE

---

## MMIOM\_DELETE Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_DELETE Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_DELETE.

---

## MMIOM\_DELETE Parameter - IParam1

**IParam1** ([LONG](#))  
Starting position for deletions.

---

## MMIOM\_DELETE Parameter - IParam2

**IParam2** ([LONG](#))  
Length of information to be deleted.

---

## MMIOM\_DELETE Return Value - rc

**rc** ([ULONG](#))

Return codes indicating success or failure:

MMIO\_SUCCESS  
The information was removed from the file sucessfully.

MMIO\_ERROR  
An error code is returned.

-----

## MMIOM\_DELETE - Description

This message is sent to an I/O procedure to request that information be removed from the file.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_DELETE.

**IParam1** ([LONG](#))  
Starting position for deletions.

**IParam2** ([LONG](#))  
Length of information to be deleted.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO\_SUCCESS  
The information was removed from the file sucessfully.

MMIO\_ERROR  
An error code is returned.

-----

## MMIOM\_DELETE - Topics

Select an item:

[Description](#)

[Returns](#)

[Glossary](#)

-----

## MMIOM\_ENDINSERT

-----

## MMIOM\_ENDINSERT Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_ENDINSERT Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_ENDINSERT.

---

## MMIOM\_ENDINSERT Parameter - IParam1

**IParam1** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_ENDINSERT Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_ENDINSERT Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The file is in overwrite mode.
MMIO_ERROR	An error code is returned.

---

## MMIOM\_ENDINSERT - Description

This message is sent to an I/O procedure to request that all subsequent [mmioWrite](#) calls overwrite data at the current seek position rather than inserting the data. This is the default mode of operation for an I/O procedure.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_ENDINSERT.

**IParam1** ([LONG](#))  
This parameter is not used.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The file is in overwrite mode.
MMIO_ERROR	An error code is returned.

---

## MMIOM\_ENDINSERT - Topics

Select an item:

- [Description](#)
- [Returns](#)
- [Glossary](#)

---

## MMIOM\_ENDRECORD

---

### MMIOM\_ENDRECORD Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_ENDRECORD Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_ENDRECORD.

# MMIOM\_ENDRECORD Parameter - IParam1

**IParam1** ([LONG](#))  
This parameter is not used.

-----

# MMIOM\_ENDRECORD Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

-----

# MMIOM\_ENDRECORD Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The request to the IOProc was successful.
MMIO_ERROR	An error code is returned.

-----

# MMIOM\_ENDRECORD - Description

This message is sent to an I/O procedure to indicate that the logical record operation has ended and internal I/O procedure data structures should be updated if necessary.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_ENDRECORD.

**IParam1** ([LONG](#))  
This parameter is not used.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The request to the IOProc was successful.
--------------	---

MMIO\_ERROR

An error code is returned.

-----

## MMIOM\_ENDRECORD - Topics

Select an item:

[Description](#)

[Returns](#)

[Glossary](#)

-----

## MMIOM\_ENDSTREAM

-----

## MMIOM\_ENDSTREAM Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

-----

## MMIOM\_ENDSTREAM Parameter - usMsg

**usMsg** ([USHORT](#))

Set to MMIOM\_ENDSTREAM.

-----

## MMIOM\_ENDSTREAM Parameter - IParam1

**IParam1** ([LONG](#))

This parameter is not used.

-----

## MMIOM\_ENDSTREAM Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_ENDSTREAM Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The file is in streaming mode.
MMIO_UNSUPPORTED_MESSAGE	I/O procedure does not support this message.
MMIO_ERROR	Streaming was unsuccessful due to other errors.

---

## MMIOM\_ENDSTREAM - Description

This message deactivates the quality of service for network I/O.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_ENDSTREAM.

**IParam1** ([LONG](#))  
This parameter is not used.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The file is in streaming mode.
MMIO_UNSUPPORTED_MESSAGE	I/O procedure does not support this message.
MMIO_ERROR	Streaming was unsuccessful due to other errors.

---

## MMIOM\_ENDSTREAM - Topics

Select an item:  
[Description](#)



---

# MMIOM\_GETCF

---

## MMIOM\_GETCF Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_GETCF Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_GETCF.

---

## MMIOM\_GETCF Parameter - IParam1

**IParam1** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_GETCF Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_GETCF Return Value - rc

**rc** ([ULONG](#))

- Returns a handle (type HMMCF) to the CTOC table that contains the element associated with *hmmio*.
- On error, NULL is returned.

-----

## MMIOM\_GETCF - Description

This message can be sent by an application to request the return of a handle *hmmcf* to the RIFF compound file that contains the element associated with *hmmio*.

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))

Set to MMIOM\_GETCF.

**IParam1** ([LONG](#))

This parameter is not used.

**IParam2** ([LONG](#))

This parameter is not used.

**rc** ([ULONG](#))

- Returns a handle (type HMMCF) to the CTOC table that contains the element associated with *hmmio*.
- On error, NULL is returned.

-----

## MMIOM\_GETCF - Topics

Select an item:

[Description](#)

[Returns](#)

[Glossary](#)

-----

## MMIOM\_GETCFENTRY

-----

## MMIOM\_GETCFENTRY Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_GETCFENTRY Parameter - usMsg

**usMsg** ([USHORT](#))

Set to MMIOM\_GETCFENTRY.

---

## MMIOM\_GETCFENTRY Parameter - IParam1

**IParam1** ([LONG](#))

A pointer to a user buffer ([MMCTOCENTRY](#)) that the CTOC entry will be read to. The user needs to allocate a large enough buffer to allow for the variable length name field and a possible extra entry field.

---

## MMIOM\_GETCFENTRY Parameter - IParam2

**IParam2** ([LONG](#))

This parameter is not used.

---

## MMIOM\_GETCFENTRY Return Value - rc

**rc** ([ULONG](#))

Return codes indicating success or type of failure:

MMIO\_CF\_SUCCESS

User buffer is updated with CTOC entry.

MMIO\_CF\_FAILURE

Error occurred; user buffer was not updated.

---

## MMIOM\_GETCFENTRY - Description

This message can be sent by an application to request a CTOC entry for an element. The element is associated with *hmmio*.

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_GETCFENTRY.

**IParam1** ([LONG](#))  
A pointer to a user buffer ([MMCTOCENTRY](#)) that the CTOC entry will be read to. The user needs to allocate a large enough buffer to allow for the variable length name field and a possible extra entry field.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or type of failure:

MMIO\_CF\_SUCCESS  
User buffer is updated with CTOC entry.

MMIO\_CF\_FAILURE  
Error occurred; user buffer was not updated.

-----

## MMIOM\_GETCFENTRY - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Glossary](#)

-----

## MMIOM\_GETFORMATINFO

-----

### MMIOM\_GETFORMATINFO Parameter - ulReserved

**ulReserved** ([ULONG](#))  
Not used. No [MMIOINFO](#) block used is required for this message.

-----

### MMIOM\_GETFORMATINFO Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_GETFORMATINFO.

-----

### MMIOM\_GETFORMATINFO Parameter - IParam1

**IParam1 (LONG)**  
A pointer to an **MMFORMATINFO** structure. The FOURCC, media type, and related information will be written into this structure by the IOProc.

-----

## MMIOM\_GETFORMATINFO Parameter - IParam2

**IParam2 (LONG)**  
This parameter is not used.

-----

## MMIOM\_GETFORMATINFO Return Value - rc

**rc (ULONG)**  
Return codes indicating success or failure:

MMIO\_SUCCESS  
The specified process installed successfully.

MMIO\_ERROR  
The installation process failed; an error code is returned.

-----

## MMIOM\_GETFORMATINFO - Description

This message requests the IOProc to return an **MMFORMATINFO** structure containing the FOURCC of the format, the capabilities of the IOProc, and other information. This message is used to initialize **MMFORMATINFO** structures internally maintained by the MMIO Manager and is issued by MMIO when installing IOProcs. This message does not require that any file be opened or referenced.

**ulReserved (ULONG)**  
Not used. No **MMIOINFO** block used is required for this message.

**usMsg (USHORT)**  
Set to MMIOM\_GETFORMATINFO.

**IParam1 (LONG)**  
A pointer to an **MMFORMATINFO** structure. The FOURCC, media type, and related information will be written into this structure by the IOProc.

**IParam2 (LONG)**  
This parameter is not used.

**rc (ULONG)**  
Return codes indicating success or failure:

MMIO\_SUCCESS  
The specified process installed successfully.

MMIO\_ERROR

The installation process failed; an error code is returned.

---

## MMIOM\_GETFORMATINFO - Topics

Select an item:

[Description](#)

[Returns](#)

[Glossary](#)

---

## MMIOM\_GETFORMATNAME

---

### MMIOM\_GETFORMATNAME Parameter - ulReserved

**ulReserved** ([ULONG](#))

Not used. No [MMIOINFO](#) block is required for this message.

---

### MMIOM\_GETFORMATNAME Parameter - usMsg

**usMsg** ([USHORT](#))

Set to MMIOM\_GETFORMATNAME.

---

### MMIOM\_GETFORMATNAME Parameter - IParam1

**IParam1** ([LONG](#))

A pointer to a [PSZ](#) buffer that contains the format name to be returned.

---

### MMIOM\_GETFORMATNAME Parameter - IParam2

**IParam2** ([LONG](#))

The expected size of the format name. The buffer passed in *//Param1* must accommodate this size. If the format name is larger than the specified size, it is truncated to *//Param2* bytes.

-----

# MMIOM\_GETFORMATNAME Return Value - rc

rc (ULONG)

- Upon successful completion, the number of bytes read into the buffer (size of format name) is returned.
- For a failure, 0 is returned.

-----

# MMIOM\_GETFORMATNAME - Description

This message requests the format name from the IOProc and will be used by MMIO when processing [mmioGetFormatName](#). This message does not require that any file be opened or referenced.

**ulReserved** (ULONG)  
Not used. No [MMIOINFO](#) block is required for this message.

**usMsg** (USHORT)  
Set to MMIOM\_GETFORMATNAME.

**IParam1** (LONG)  
A pointer to a [PSZ](#) buffer that contains the format name to be returned.

**IParam2** (LONG)  
The expected size of the format name. The buffer passed in *//Param1* must accommodate this size. If the format name is larger than the specified size, it is truncated to *//Param2* bytes.

rc (ULONG)

- Upon successful completion, the number of bytes read into the buffer (size of format name) is returned.
- For a failure, 0 is returned.

-----

# MMIOM\_GETFORMATNAME - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Glossary](#)

-----

# MMIOM\_GETHEADER

-----

## MMIOM\_GETHEADER Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

-----

## MMIOM\_GETHEADER Parameter - usMsg

**usMsg** ([USHORT](#))

Set to MMIOM\_GETHEADER.

-----

## MMIOM\_GETHEADER Parameter - IParam1

**IParam1** ([LONG](#))

Pointer to a file-specific header structure that will contain information provided by the IOProc.

-----

## MMIOM\_GETHEADER Parameter - IParam2

**IParam2** ([LONG](#))

Length in bytes of the structure supplied in *IParam1*.

-----

## MMIOM\_GETHEADER Return Value - rc

**rc** ([ULONG](#))

- Upon successful completion, the number of bytes copied to the header structure.
- For a failure, 0 is returned.
- If the length passed in was not enough to hold the header, MMIOERR\_INVALID\_BUFFER\_LENGTH is set in *ulErrorRet*.
- If the header is bad, MMIOERR\_INVALID\_STRUCTURE is set in *ulErrorRet*.



---

## MMIOM\_GETHEADER - Description

This message requests the IOProc to return media-specific information about the current file or file element. This would include such data as resolution and colors for images, and duration and sample rate for audio. Header translation is expected to be performed when specified.

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))

Set to MMIOM\_GETHEADER.

**IParam1** ([LONG](#))

Pointer to a file-specific header structure that will contain information provided by the IOProc.

**IParam2** ([LONG](#))

Length in bytes of the structure supplied in *IParam1*.

**rc** ([ULONG](#))

- Upon successful completion, the number of bytes copied to the header structure.
- For a failure, 0 is returned.
- If the length passed in was not enough to hold the header, MMIOERR\_INVALID\_BUFFER\_LENGTH is set in *ulErrorRet*.
- If the header is bad, MMIOERR\_INVALID\_STRUCTURE is set in *ulErrorRet*.

---

## MMIOM\_GETHEADER - Remarks

This message requires that a valid *hmmio* handle be returned from a successful call to [MMIOM\\_OPEN](#)

See [mmioGetHeader](#) for more information about header information.

Examples of some header structures that *IParam1* might point to are:

- [MMIMAGEHEADER](#) (includes structure length, content, size, color type, and other information, including space for a 256-color palette)
- [MMAUDIOHEADER](#) (includes such data as structure length, content, samples per second, and sample size)
- [MMMIDIHEADER](#) (contains all pertinent information about the MIDI)
- [MMMOVIEHEADER](#)
- [MMVIDEOHEADER](#)

---

## MMIOM\_GETHEADER - Topics

Select an item:

[Description](#)

---

# MMIOM\_IDENTIFYFILE

---

## MMIOM\_IDENTIFYFILE Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
An [MMIOINFO](#) block is optional for this message.

---

## MMIOM\_IDENTIFYFILE Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_IDENTIFYFILE.

---

## MMIOM\_IDENTIFYFILE Parameter - IParam1

**IParam1** ([LONG](#))  
An optional pointer to a string, *pszFileName*, containing the name of the file to be evaluated. This string must be provided by the sender of the message. It should follow the form defined by the [mmioOpen](#) function.

---

## MMIOM\_IDENTIFYFILE Parameter - IParam2

**IParam2** ([LONG](#))  
An *hmmio* file handle must be specified. The IOProc uses this handle to read from the file instead of opening the file again.

---

## MMIOM\_IDENTIFYFILE Return Value - rc

**rc** ([ULONG](#))

Return codes indicating success or failure:

MMIO\_SUCCESS

The file in the format of, and supported by this IOProc.

MMIO\_ERROR

The file cannot be supported by this IOProc.

-----

## MMIOM\_IDENTIFYFILE - Description

This message attempts to determine if any IOProc can process a file or file element. It does not require that the file be already opened by [MMIOM\\_OPEN](#).

**pmmioinfo** ([PMMIOINFO](#))

An [MMIOINFO](#) block is optional for this message.

**usMsg** ([USHORT](#))

Set to MMIOM\_IDENTIFYFILE.

**IParam1** ([LONG](#))

An optional pointer to a string, *pszFileName*, containing the name of the file to be evaluated. This string must be provided by the sender of the message. It should follow the form defined by the [mmioOpen](#) function.

**IParam2** ([LONG](#))

An *hmmio* file handle must be specified. The IOProc uses this handle to read from the file instead of opening the file again.

**rc** ([ULONG](#))

Return codes indicating success or failure:

MMIO\_SUCCESS

The file in the format of, and supported by this IOProc.

MMIO\_ERROR

The file cannot be supported by this IOProc.

-----

## MMIOM\_IDENTIFYFILE - Remarks

The IOProc should not depend solely on the extension of the file, but should actually interrogate the file contents, such as header information, to ensure support.

MMIOM\_IDENTIFYFILE does not require the file to have been opened. The IOProc will normally use [mmioOpen](#) and related calls to determine if the file is of the correct form.

Normally [MMIOINFO](#) is NULL unless an element given is not valid.

-----

## MMIOM\_IDENTIFYFILE - Topics

Select an item:

[Description](#)

[Returns](#)

---

# MMIOM\_MULTITRACKREAD

---

## MMIOM\_MULTITRACKREAD Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_MULTITRACKREAD Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_MULTITRACKREAD.

---

## MMIOM\_MULTITRACKREAD Parameter - IParam1

**IParam1** ([LONG](#))  
Pointer to the [MMMULTITRACKREAD](#) structure.

---

## MMIOM\_MULTITRACKREAD Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_MULTITRACKREAD Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO\_SUCCESS  
The association is completed successfully.

MMIO\_ERROR  
An error code is returned.

---

## MMIOM\_MULTITRACKREAD - Description

This message is sent to the I/O procedure to request that data be read from one or more tracks from a multiple-track file.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_MULTITRACKREAD.

**IParam1** ([LONG](#))  
Pointer to the [MMMULTITRACKREAD](#) structure.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO\_SUCCESS  
The association is completed successfully.

MMIO\_ERROR  
An error code is returned.

---

## MMIOM\_MULTITRACKREAD - Topics

Select an item:

- [Description](#)
- [Returns](#)
- [Glossary](#)

---

## MMIOM\_MULTITRACKWRITE

---

## MMIOM\_MULTITRACKWRITE Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

-----

## MMIOM\_MULTITRACKWRITE Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_MULTITRACKWRITE.

-----

## MMIOM\_MULTITRACKWRITE Parameter - IParam1

**IParam1** ([LONG](#))  
Pointer to the [MMMULTITRACKWRITE](#) structure.

-----

## MMIOM\_MULTITRACKWRITE Parameter - IParam2

**IParam2** ([LONG](#))  
Not used.

-----

## MMIOM\_MULTITRACKWRITE Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The association is completed successfully.
MMIO_ERROR	An error code is returned.

-----

## MMIOM\_MULTITRACKWRITE - Description

This message is sent to the I/O procedure to request that data for one or more tracks be written to a multiple-track file. The data for the tracks will be interleaved by the I/O procedure.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_MULTITRACKWRITE.

**IParam1** ([LONG](#))  
Pointer to the [MMMULTITRACKWRITE](#) structure.

**IParam2** ([LONG](#))  
Not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The association is completed successfully.
MMIO_ERROR	An error code is returned.

---

## MMIOM\_MULTITRACKWRITE - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Glossary](#)

---

## MMIOM\_OPEN

---

### MMIOM\_OPEN Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_OPEN Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_OPEN.

---

### MMIOM\_OPEN Parameter - IParam1

**IParam1** ([LONG](#))  
The *pszFileName* argument of [mmioOpen](#).

---

## MMIOM\_OPEN Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_OPEN Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or type of failure:

**MMIO\_SUCCESS**  
The specified file is opened or deleted successfully. Otherwise, either an OS/2 error code or an MMIO error code is returned. See [Return Codes](#) for a description of the MMIO Manager error codes.

---

## MMIOM\_OPEN - Description

This message is sent to an IOProc by [mmioOpen](#) to request that a file be opened or deleted.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_OPEN.

**IParam1** ([LONG](#))  
The *pszFileName* argument of [mmioOpen](#).

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or type of failure:

**MMIO\_SUCCESS**  
The specified file is opened or deleted successfully. Otherwise, either an OS/2 error code or an MMIO error code is returned. See [Return Codes](#) for a description of the MMIO Manager error codes.

---

## MMIOM\_OPEN - Remarks



The *IDiskOffset* field of the [MMIOINFO](#) structure is initialized to 0 by [mmioOpen](#) before MMIOM\_OPEN is called. If this value is incorrect, the IOProc must correct it.

See [mmioOpen](#) for a description of the file *ulFlags* field of *pmmioinfo* (which is passed to [mmioOpen](#) as *ulOpenFlags*). In particular, if the MMIO\_DELETE flag is present, the file must be deleted.

---

## MMIOM\_OPEN - Topics

Select an item:

[Description](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

---

## MMIOM\_PASTE

---

### MMIOM\_PASTE Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_PASTE Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_PASTE.

---

### MMIOM\_PASTE Parameter - IParam1

**IParam1** ([LONG](#))  
A pointer to an [MMIO\\_EDIT\\_PARMS](#) structure.

---

### MMIOM\_PASTE Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_PASTE Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The request was successful.
MMIO_ERROR	An error code is returned.

---

## MMIOM\_PASTE - Description

This message is sent to an I/O procedure to request that data from the clipboard be inserted into the file at the position specified.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_PASTE.

**IParam1** ([LONG](#))  
A pointer to an [MMIO\\_EDIT\\_PARMS](#) structure.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The request was successful.
MMIO_ERROR	An error code is returned.

---

## MMIOM\_PASTE - Remarks

Data from the clipboard is inserted into the file at the media position immediately before the *ulStartTime* position. ([MMIO\\_EDIT\\_PARMS](#) structure passed in the *IParam1* parameter). If the *ulDuration* field of the same structure is not zero, a delete operation is performed for the specified range prior to the insertion. After completion of this operation, the media position will be immediately after the pasted data.

If the *ulDuration* is zero, no deletion of data will take place before the pasting of clipboard data into the file.

---

# MMIOM\_PASTE - Topics

- Select an item:
- Description
  - Returns
  - Remarks
  - Glossary

---

## MMIOM\_QUERYHEADERLENGTH

---

### MMIOM\_QUERYHEADERLENGTH Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_QUERYHEADERLENGTH Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_QUERYHEADERLENGTH.

---

### MMIOM\_QUERYHEADERLENGTH Parameter - IParam1

**IParam1** ([LONG](#))  
This parameter is not used.

---

### MMIOM\_QUERYHEADERLENGTH Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_QUERYHEADERLENGTH Return Value - rc

rc ([ULONG](#))

- Upon successful completion, the size of the header, in bytes, is returned.
  - For a failure, 0 is returned.
- 

## MMIOM\_QUERYHEADERLENGTH - Description

This message requests the IOProc to return the size of the header for the current file or file element. The file was opened with [mmioOpen](#).

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))

Set to MMIOM\_QUERYHEADERLENGTH.

**IParam1** ([LONG](#))

This parameter is not used.

**IParam2** ([LONG](#))

This parameter is not used.

rc ([ULONG](#))

- Upon successful completion, the size of the header, in bytes, is returned.
  - For a failure, 0 is returned.
- 

## MMIOM\_QUERYHEADERLENGTH - Remarks

The IOProc is expected to return the standard presentation format header length when header translation is specified. The following are examples of some structures that the preceding parameters might use. These are the standard presentation format header structures.

- [MMIMAGEHEADER](#), includes structure length, content, size, color type and other information, including space for a 256-color palette.
  - [MMAUDIOHEADER](#), includes structure length, content, samples per second, and sample size.
  - [MMMIDIHEADER](#), contains all pertinent information about the MIDI.
  - [MMMOVIEHEADER](#) contains information about the movie.
  - [MMVIDEOHEADER](#) contains information about the digital video data.
-

# MMIOM\_QUERYHEADERLENGTH - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Remarks](#)  
[Glossary](#)

---

## MMIOM\_QUERYIMAGE

---

### MMIOM\_QUERYIMAGE Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_QUERYIMAGE Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_QUERYIMAGE.

---

### MMIOM\_QUERYIMAGE Parameter - pulParam1

**pulParam1** ([PULONG](#))  
Pointer to a ULONG that will contain the current image index upon return. Image indexes are zero-based.

---

### MMIOM\_QUERYIMAGE Parameter - lParam2

**lParam2** ([LONG](#))  
This parameter is not used.

---

# MMIOM\_QUERYIMAGE Return Value - rc

**rc** ([ULONG](#))

Return codes indicating success or failure.

MMIO\_SUCCESS

The request was successful.

MMIO\_ERROR

An error code is returned.

MMIOERR\_UNSUPPORTED\_FUNCTION

This command is not supported by this image IOProc, so it should be interpreted as having the 0th index image selected.

-----

## MMIOM\_QUERYIMAGE - Description

This message is sent to an image IOProc to determine the currently selected image index in the image file.

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))

Set to MMIOM\_QUERYIMAGE.

**pulParam1** ([PULONG](#))

Pointer to a ULONG that will contain the current image index upon return. Image indexes are zero-based.

**IPParam2** ([LONG](#))

This parameter is not used.

**rc** ([ULONG](#))

Return codes indicating success or failure.

MMIO\_SUCCESS

The request was successful.

MMIO\_ERROR

An error code is returned.

MMIOERR\_UNSUPPORTED\_FUNCTION

This command is not supported by this image IOProc, so it should be interpreted as having the 0th index image selected.

-----

## MMIOM\_QUERYIMAGE - Related Messages

- [MMIOM\\_QUERYIMAGECOUNT](#)
- [MMIOM\\_SETIMAGE](#)

---

## MMIOM\_QUERYIMAGE - Topics

Select an item:

[Description](#)

[Returns](#)

[Related Messages](#)

[Glossary](#)

---

## MMIOM\_QUERYIMAGECOUNT

---

### MMIOM\_QUERYIMAGECOUNT Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_QUERYIMAGECOUNT Parameter - usMsg

**usMsg** ([USHORT](#))

Set to MMIOM\_QUERYIMAGECOUNT.

---

### MMIOM\_QUERYIMAGECOUNT Parameter - pulParam1

**pulParam1** ([PULONG](#))

Pointer to a ULONG that will contain the count of images in this file when the command completes.

---

### MMIOM\_QUERYIMAGECOUNT Parameter - IParam2

**IParam2** ([LONG](#))

This parameter is not used.

---

## MMIOM\_QUERYIMAGECOUNT Return Value - rc

**rc** ([ULONG](#))

Return codes indicating success or failure:

MMIO\_SUCCESS

The request was successful.

MMIO\_ERROR

An error code is returned.

MMIOERR\_UNSUPPORTED\_FUNCTION

This command is not supported by this image IOProc, so it should be interpreted as supporting only one image.

---

## MMIOM\_QUERYIMAGECOUNT - Description

This message is sent to the IOProc to determine the number of images that are stored in the image file.

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))

Set to MMIOM\_QUERYIMAGECOUNT.

**pulParam1** ([PULONG](#))

Pointer to a ULONG that will contain the count of images in this file when the command completes.

**IPParam2** ([LONG](#))

This parameter is not used.

**rc** ([ULONG](#))

Return codes indicating success or failure:

MMIO\_SUCCESS

The request was successful.

MMIO\_ERROR

An error code is returned.

MMIOERR\_UNSUPPORTED\_FUNCTION

This command is not supported by this image IOProc, so it should be interpreted as supporting only one image.

---

## MMIOM\_QUERYIMAGECOUNT - Related Messages

- [MMIOM\\_QUERYIMAGE](#)
  - [MMIOM\\_SETIMAGE](#)
-



# MMIOM\_QUERYIMAGECOUNT - Topics

- Select an item:
- [Description](#)
  - [Returns](#)
  - [Related Messages](#)
  - [Glossary](#)

---

## MMIOM\_READ

---

### MMIOM\_READ Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_READ Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_READ.

---

### MMIOM\_READ Parameter - IParam1

**IParam1** ([LONG](#))  
A (PSZ) pointer to the buffer to read to.

---

### MMIOM\_READ Parameter - IParam2

**IParam2** ([LONG](#))  
The number of bytes to read.

---

# MMIOM\_READ Return Value - rc

**rc** ([ULONG](#))  
Returns the number of bytes actually read. Returns 0L if no more bytes can be read.

**MMIO\_ERROR**  
READ was not successful.

## MMIOM\_READ - Description

This message is sent to an IOProc by [mmioRead](#) to request that bytes be read from an open file. Data translation is expected during message processing when specified.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_READ.

**IParam1** ([LONG](#))  
A (PSZ) pointer to the buffer to read to.

**IParam2** ([LONG](#))  
The number of bytes to read.

**rc** ([ULONG](#))  
Returns the number of bytes actually read. Returns 0L if no more bytes can be read.

**MMIO\_ERROR**  
READ was not successful.

## MMIOM\_READ - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Glossary](#)

## MMIOM\_REDO

## MMIOM\_REDO Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_REDO Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_REDO.

---

## MMIOM\_REDO Parameter - IParam1

**IParam1** ([LONG](#))  
Not used.

---

## MMIOM\_REDO Parameter - IParam2

**IParam2** ([LONG](#))  
Not used.

---

## MMIOM\_REDO Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The request to the IOProc was successful.
MMIO_ERROR	An error code is returned.

---

## MMIOM\_REDO - Description

This message is sent to an I/O procedure to request the last logical action which was undone by [MMIOM\\_UNDO](#) be redone.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_REDO.

**IParam1** ([LONG](#))  
Not used.

**IParam2** ([LONG](#))  
Not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

[MMIO\\_SUCCESS](#)  
The request to the IOProc was successful.

[MMIO\\_ERROR](#)  
An error code is returned.

---

## MMIOM\_REDO - Topics

Select an item:

[Description](#)

[Returns](#)

[Glossary](#)

---

## MMIOM\_SAVE

---

## MMIOM\_SAVE Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_SAVE Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_SAVE.

# MMIOM\_SAVE Parameter - IParam1

**IParam1** ([LONG](#))  
Optional *pszFileName* parameter.

-----

# MMIOM\_SAVE Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

-----

# MMIOM\_SAVE Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The file has been saved.
MMIO_ERROR	An error code is returned.

-----

# MMIOM\_SAVE - Description

This message is sent to an I/O procedure to request temporary changes in a file be made permanent. A new file name can be supplied to save the changes.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_SAVE.

**IParam1** ([LONG](#))  
Optional *pszFileName* parameter.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The file has been saved.
--------------	--------------------------

MMIO\_ERROR  
An error code is returned.

---

# MMIOM\_SAVE - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Glossary](#)

---

# MMIOM\_SEEK

---

# MMIOM\_SEEK Parameter - pmmioinfo

pmmioinfo ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

# MMIOM\_SEEK Parameter - usMsg

usMsg ([USHORT](#))  
Set to MMIOM\_SEEK.

---

# MMIOM\_SEEK Parameter - IParam1

IParam1 ([LONG](#))  
The signed distance (offset) to move, specified as bytes.

---

# MMIOM\_SEEK Parameter - IParam2

**IParam2 (LONG)**

Contains one of the following values:

**SEEK\_SET**

Moves the file pointer to be *IParam1* bytes from the beginning of the file.

**MMIO\_SEEK\_IFRAME**

This results in a seek to the nearest I-frame preceding the position determined by *IParam1* and the other *IParam2* flags.

**SEEK\_CUR**

Moves the file position to be *IParam1* bytes from the current position. *IParam1* can be positive or negative.

**SEEK\_END**

Moves the file position to be *IParam1* bytes from the end of the file. *IParam1* can be positive or negative.

-----

## MMIOM\_SEEK Return Value - rc

**rc (ULONG)**

Returns the new file position or MMIO\_ERROR if the seek was not successful.

-----

## MMIOM\_SEEK - Description

This message is sent to an IOProc by [mmioSeek](#) to request that the current file position (as maintained by the I/O procedure and stored in the *DiskOffset* field of the [MMIOINFO](#) structure) be moved.

**pmmioinfo (PMMIOINFO)**

A pointer to an [MMIOINFO](#) data structure.

**usMsg (USHORT)**

Set to MMIOM\_SEEK.

**IParam1 (LONG)**

The signed distance (offset) to move, specified as bytes.

**IParam2 (LONG)**

Contains one of the following values:

**SEEK\_SET**

Moves the file pointer to be *IParam1* bytes from the beginning of the file.

**MMIO\_SEEK\_IFRAME**

This results in a seek to the nearest I-frame preceding the position determined by *IParam1* and the other *IParam2* flags.

**SEEK\_CUR**

Moves the file position to be *IParam1* bytes from the current position. *IParam1* can be positive or negative.

**SEEK\_END**

Moves the file position to be *IParam1* bytes from the end of the file. *IParam1* can be positive or negative.

**rc (ULONG)**

Returns the new file position or MMIO\_ERROR if the seek was not successful.

---

## MMIOM\_SEEK - Topics

Select an item:

[Description](#)

[Returns](#)

[Glossary](#)

---

## MMIOM\_SEEKBYTIME

---

### MMIOM\_SEEKBYTIME Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_SEEKBYTIME Parameter - usMsg

**usMsg** ([USHORT](#))

Set to MMIOM\_SEEKBYTIME.

---

### MMIOM\_SEEKBYTIME Parameter - IParam1

**IParam1** ([LONG](#))

Contains the signed offset to move, specified in MMTIME.

---

### MMIOM\_SEEKBYTIME Parameter - IParam2

**IParam2** ([LONG](#))

Specifies how the *IParam1* parameter is interpreted:

SEEK\_SET

Seek to an absolute (time units from the beginning of the file) seek position specified in *IParam1*. This is the



default.

SEEK\_CUR

Seek to a relative (time units from the current file position) seek position specified in *IParam1*. (The value of *IParam1* can be positive or negative.)

SEEK\_END

Seek to *IParam1* time units from the end of the file. (The value of *IParam1* can be positive or negative.)

-----

## MMIOM\_SEEKBYTIME Return Value - rc

rc (ULONG)

Returns the new file position or MMIO\_ERROR if the seek was not successful.

-----

## MMIOM\_SEEKBYTIME - Description

This message is sent to an I/O procedure by [mmioSendMessage](#) and requests the file position (as maintained by the I/O procedure and stored in the *ILDiskOffset* field of the [MMIOINFO](#) structure) be moved relative to some unit of time understood by the I/O procedure.

pmmioinfo ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

usMsg ([USHORT](#))

Set to MMIOM\_SEEKBYTIME.

IParam1 ([LONG](#))

Contains the signed offset to move, specified in MMTIME.

IParam2 ([LONG](#))

Specifies how the *IParam1* parameter is interpreted:

SEEK\_SET

Seek to an absolute (time units from the beginning of the file) seek position specified in *IParam1*. This is the default.

SEEK\_CUR

Seek to a relative (time units from the current file position) seek position specified in *IParam1*. (The value of *IParam1* can be positive or negative.)

SEEK\_END

Seek to *IParam1* time units from the end of the file. (The value of *IParam1* can be positive or negative.)

rc (ULONG)

Returns the new file position or MMIO\_ERROR if the seek was not successful.

-----

## MMIOM\_SEEKBYTIME - Remarks

Currently, time units are expressed in terms of MMTIME time units (1/3000 of a second). Only some I/O procedures support this.

---

# MMIOM\_SEEKBYTIME - Topics

- Select an item:
- Description
  - Returns
  - Remarks
  - Glossary

---

## MMIOM\_SET

---

### MMIOM\_SET Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_SET Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_SET.

---

### MMIOM\_SET Parameter - IParam1

**IParam1** ([LONG](#))  
Pointer to the [MMEXTENDINFO](#) structure.

---

### MMIOM\_SET Parameter - IParam2

**IParam2** ([LONG](#))  
Contains one of the following flags:

- MMIO\_SET\_EXTENDEDINFO

Sets extended information.

MMIO\_QUERY\_EXTENDEDINFO\_BASE

Query only the information of [MMEXTENDINFO](#) structure.

MMIO\_QUERY\_EXTENDEDINFO\_ALL

Query all extended information including the CODEC associated information.

-----

## MMIOM\_SET Return Value - rc

**rc** ([ULONG](#))

Return codes indicating success or failure:

MMIO\_SUCCESS

The set is completed successfully.

MMIO\_ERROR

An error code is returned.

-----

## MMIOM\_SET - Description

This message is sent to the IOProc to set or query extended file information.

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))

Set to MMIOM\_SET.

**IParam1** ([LONG](#))

Pointer to the [MMEXTENDINFO](#) structure.

**IParam2** ([LONG](#))

Contains one of the following flags:

MMIO\_SET\_EXTENDEDINFO

Sets extended information.

MMIO\_QUERY\_EXTENDEDINFO\_BASE

Query only the information of [MMEXTENDINFO](#) structure.

MMIO\_QUERY\_EXTENDEDINFO\_ALL

Query all extended information including the CODEC associated information.

**rc** ([ULONG](#))

Return codes indicating success or failure:

MMIO\_SUCCESS

The set is completed successfully.

MMIO\_ERROR

An error code is returned.

-----

# MMIOM\_SET - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Glossary](#)

---

## MMIOM\_SETHEADER

---

### MMIOM\_SETHEADER Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_SETHEADER Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_SETHEADER.

---

### MMIOM\_SETHEADER Parameter - IParam1

**IParam1** ([LONG](#))  
Pointer to a header buffer that contains the data to be written by the IOProc to the header of the file element. The application is responsible for creating and completing the data structure.

---

### MMIOM\_SETHEADER Parameter - IParam2

**IParam2** ([LONG](#))  
Length in bytes of the structure supplied in *IParam1*.

---

# MMIOM\_SETHHEADER Return Value - rc

rc (ULONG)

- Upon successful completion, the number of bytes written is returned.
- For a failure, 0 is returned.
- If the header is invalid, MMIOERR\_INVALID\_STRUCTURE is set in *ulErrorRet*.

-----

## MMIOM\_SETHHEADER - Description

This message requests the IOProc to use the media-specific information when accepting data about, and writing to, the current file or file element. This would include such data as resolution and colors for images, and duration and sample rate for audio. Header translation is expected to be performed when specified.

pmmioinfo (PMMIOINFO)

A pointer to an MMIOINFO data structure.

usMsg (USHORT)

Set to MMIOM\_SETHHEADER.

IParam1 (LONG)

Pointer to a header buffer that contains the data to be written by the IOProc to the header of the file element. The application is responsible for creating and completing the data structure.

IParam2 (LONG)

Length in bytes of the structure supplied in *IParam1*.

rc (ULONG)

- Upon successful completion, the number of bytes written is returned.
- For a failure, 0 is returned.
- If the header is invalid, MMIOERR\_INVALID\_STRUCTURE is set in *ulErrorRet*.

-----

## MMIOM\_SETHHEADER - Remarks

This message requires that a valid *hmmio* handle be returned from a successful call to [MMIOM\\_OPEN](#).

See [mmioSetHeader](#) for more information about header information.

The contents of the header must represent the structure that is expected by the IOProc. It does not represent the way in which data is saved by the IOProc because the IOProc might translate the data in some manner.

Examples of the standard presentation format headers that *IParam1* might point to are:

- [MMIMAGEHEADER](#), includes structure length, content, size, color type, and other information, including space for a 256-color

palette.

- [MMAUDIOHEADER](#), includes such data as structure length, content, samples per second, and sample size.
- [MMMIDIHEADER](#), contains all pertinent information about the MIDI.
- [MMMOVIEHEADER](#)
- [MMVIDEOHEADER](#)

-----

## MMIOM\_SETHEADER - Topics

Select an item:

[Description](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

-----

## MMIOM\_SETIMAGE

-----

## MMIOM\_SETIMAGE Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))

Pointer to an [MMIOINFO](#) data structure.

-----

## MMIOM\_SETIMAGE Parameter - usMsg

**usMsg** ([USHORT](#))

Set to MMIOM\_SETIMAGE.

-----

## MMIOM\_SETIMAGE Parameter - ulParam1

**ulParam1** ([ULONG](#))

A ULONG containing the new image index. If the index is less than the count, then an existing image will be addressed. If the index is equal to the count, then a new image will be created when [mmioSetHeader](#) is called. Indexes greater than the count will generate an error.

---

## MMIOM\_SETIMAGE Parameter - IParam2

**IParam2** ([LONG](#))

This parameter is not used.

---

## MMIOM\_SETIMAGE Return Value - rc

**rc** ([ULONG](#))

Return codes indicating success or failure.

MMIO\_SUCCESS

The request was successful.

MMIO\_ERROR

An error code is returned.

MMIOERR\_UNSUPPORTED\_FUNCTION

The IOProc does not support multiple images.

---

## MMIOM\_SETIMAGE - Description

This message is sent to an image IOProc to select a new image index in the image file.

**pmmioinfo** ([PMMIOINFO](#))

Pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))

Set to MMIOM\_SETIMAGE.

**ulParam1** ([ULONG](#))

A ULONG containing the new image index. If the index is less than the count, then an existing image will be addressed. If the index is equal to the count, then a new image will be created when [mmioSetHeader](#) is called. Indexes greater than the count will generate an error.

**IParam2** ([LONG](#))

This parameter is not used.

**rc** ([ULONG](#))

Return codes indicating success or failure.

MMIO\_SUCCESS

The request was successful.

MMIO\_ERROR

An error code is returned.

MMIOERR\_UNSUPPORTED\_FUNCTION

The IOProc does not support multiple images.

---

# MMIOM\_SETIMAGE - Related Messages

- [MMIOM\\_QUERYIMAGECOUNT](#)
- [MMIOM\\_QUERYIMAGE](#)

---

# MMIOM\_SETIMAGE - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Related Messages](#)  
[Glossary](#)

---

# MMIOM\_STATUS

---

## MMIOM\_STATUS Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_STATUS Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_STATUS.

---

## MMIOM\_STATUS Parameter - IParam1

**IParam1** ([LONG](#))  
A pointer to an [MMIO\\_STATUS\\_PARMS](#) structure.



---

## MMIOM\_STATUS Parameter - IParam2

**IParam2** ([LONG](#))

This parameter is not used.

---

## MMIOM\_STATUS Return Value - rc

**rc** ([ULONG](#))

Return codes indicating success or failure:

MMIO\_SUCCESS

The request was successful.

MMIO\_ERROR

An error code is returned.

MMIOERR\_MISSING\_FLAG

A required flag was not supplied with the command.

MMIOERR\_INVALID\_ITEM\_FLAG

One or more of the item flags specified are invalid for this command.

---

## MMIOM\_STATUS - Description

This message is used to pass [MCI\\_STATUS](#) messages to an I/O procedure.

**pmmioinfo** ([PMMIOINFO](#))

A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))

Set to MMIOM\_STATUS.

**IParam1** ([LONG](#))

A pointer to an [MMIO\\_STATUS\\_PARMS](#) structure.

**IParam2** ([LONG](#))

This parameter is not used.

**rc** ([ULONG](#))

Return codes indicating success or failure:

MMIO\_SUCCESS

The request was successful.

MMIO\_ERROR

An error code is returned.

MMIOERR\_MISSING\_FLAG

A required flag was not supplied with the command.

MMIOERR\_INVALID\_ITEM\_FLAG  
One or more of the item flags specified are invalid for this command.

-----

## MMIOM\_STATUS - Remarks

A TRUE or FALSE value is returned in the *ulReturn* field of the [MMIO\\_STATUS\\_PARMS](#) structure depending on the status of the item. The *ulType* field of this structure gives the MCI\_FORMAT flag for the returned data when appropriate.

Item flags for the [MCI\\_STATUS](#) structure are used in the *ulItem* field of the [MMIO\\_STATUS\\_PARMS](#) structure. The following flags can be used in the *ulItem* field:

- MCI\_STATUS\_CAN\_PASTE
- MCI\_STATUS\_CLIPBOARD
- MCI\_STATUS\_CAN\_REDO
- MCI\_STATUS\_CAN\_UNDO

-----

## MMIOM\_STATUS - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Remarks](#)  
[Glossary](#)

-----

## MMIOM\_TEMPCHANGE

-----

### MMIOM\_TEMPCHANGE Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

-----

### MMIOM\_TEMPCHANGE Parameter - usMsg

**usMsg** ([USHORT](#))

Set to MMIOM\_TEMPCHANGE.

---

## MMIOM\_TEMPCHANGE Parameter - IParam1

### IParam1 (LONG)

This parameter contains a pointer to a null-terminated string which has the name of a directory where a temporary file should be created.

---

## MMIOM\_TEMPCHANGE Parameter - IParam2

### IParam2 (LONG)

This parameter is not used.

---

## MMIOM\_TEMPCHANGE Return Value - rc

### rc (ULONG)

Return codes indicating success or failure:

#### MMIO\_SUCCESS

The file is recording changes as temporary.

#### MMIO\_ERROR

An error code is returned.

---

## MMIOM\_TEMPCHANGE - Description

This message is sent to an I/O procedure to request all subsequent [mmioWrite](#) calls to an I/O procedure for a file be treated as temporary changes. The changes will not be saved to the file when closed by [mmioClose](#) unless an [MMIOM\\_SAVE](#) message is received.

### pmmioinfo (PMMIOINFO)

A pointer to an [MMIOINFO](#) data structure.

### usMsg (USHORT)

Set to MMIOM\_TEMPCHANGE.

### IParam1 (LONG)

This parameter contains a pointer to a null-terminated string which has the name of a directory where a temporary file should be created.

### IParam2 (LONG)

This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

<b>MMIO_SUCCESS</b>	The file is recording changes as temporary.
<b>MMIO_ERROR</b>	An error code is returned.

---

## MMIOM\_TEMPCHANGE - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Glossary](#)

---

## MMIOM\_UNDO

---

### MMIOM\_UNDO Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

### MMIOM\_UNDO Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_UNDO.

---

### MMIOM\_UNDO Parameter - IParam1

**IParam1** ([LONG](#))  
This parameter is not used.

---

# MMIOM\_UNDO Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_UNDO Return Value - rc

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The request to the IOProc was successful.
MMIO_ERROR	An error code is returned.

---

## MMIOM\_UNDO - Description

This message is sent to an I/O procedure to request that the last logical action ([MMIOM\\_DELETE](#), [MMIOM\\_BEGININSERT](#), [MMIOM\\_ENDINSERT](#), [MMIOM\\_UNDO](#), or [MMIOM\\_REDO](#)) be undone.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_UNDO.

**IParam1** ([LONG](#))  
This parameter is not used.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return codes indicating success or failure:

MMIO_SUCCESS	The request to the IOProc was successful.
MMIO_ERROR	An error code is returned.

---

## MMIOM\_UNDO - Topics

Select an item:

[Description](#)  
[Returns](#)  
[Glossary](#)

---

# MMIOM\_WINMSG

---

## MMIOM\_WINMSG Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_WINMSG Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_WINMSG.

---

## MMIOM\_WINMSG Parameter - IParam1

**IParam1** ([LONG](#))  
A pointer to an [MMIO\\_WINMSG](#) structure.

---

## MMIOM\_WINMSG Parameter - IParam2

**IParam2** ([LONG](#))  
This parameter is not used.

---

## MMIOM\_WINMSG Return Value - rc

**rc** ([ULONG](#))

Return code indicating success or failure.

MMIO\_SUCCESS  
The request to the IOProc was successful.

MMIO\_ERROR  
An error code is returned.

---

## MMIOM\_WINMSG - Description

This message allows an application or an MCD to pass PM messages from a window procedure to an I/O procedure. It is currently used to pass a WM\_DESTROYCLIPBOARD message to an I/O procedure for appropriate action. It is an optional message and may not be supported by an I/O procedure, therefore any errors should be ignored by the caller.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

**usMsg** ([USHORT](#))  
Set to MMIOM\_WINMSG.

**IParam1** ([LONG](#))  
A pointer to an [MMIO\\_WINMSG](#) structure.

**IParam2** ([LONG](#))  
This parameter is not used.

**rc** ([ULONG](#))  
Return code indicating success or failure.

MMIO\_SUCCESS  
The request to the IOProc was successful.

MMIO\_ERROR  
An error code is returned.

---

## MMIOM\_WINMSG - Topics

Select an item:

[Description](#)

[Returns](#)

[Glossary](#)

---

## MMIOM\_WRITE

---

## MMIOM\_WRITE Parameter - pmmioinfo

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.

---

## MMIOM\_WRITE Parameter - usMsg

**usMsg** ([USHORT](#))  
Set to MMIOM\_WRITE.

---

## MMIOM\_WRITE Parameter - IParam1

**IParam1** ([LONG](#))  
A pointer (PCHAR) to the buffer to write form.

---

## MMIOM\_WRITE Parameter - IParam2

**IParam2** ([LONG](#))  
The number of bytes to write.

---

## MMIOM\_WRITE Return Value - rc

**rc** ([ULONG](#))  
Returns the number of bytes actually written or MMIO\_ERROR if the write was not successful.

---

## MMIOM\_WRITE - Description

This message is sent to an IOProc by [mmioWrite](#) to request that bytes be written to an open file.

**pmmioinfo** ([PMMIOINFO](#))  
A pointer to an [MMIOINFO](#) data structure.



**usMsg** ([USHORT](#))  
Set to MMIOM\_WRITE.

**IParam1** ([LONG](#))  
A pointer (PCHAR) to the buffer to write form.

**IParam2** ([LONG](#))  
The number of bytes to write.

**rc** ([ULONG](#))  
Returns the number of bytes actually written or MMIO\_ERROR if the write was not successful.

-----

## MMIOM\_WRITE - Topics

Select an item:  
[Description](#)  
[Returns](#)  
[Glossary](#)

-----